内容分发网络原理与实践

Principle and Practice of CDN

唐宏　陈戈　陈步华　余媛◎编著

系统地分析了CDN基本原理与关键技术
介绍了CDN的选择依据
构建了基于开源软件的CDN系统

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

作者简介

唐　宏

硕士，毕业于华中科技大学，现任中国电信股份有限公司广州研究院数据通信研究所所长，从事多年数据通信、CDN、云计算、SDN等领域研究工作，曾获中国科学技术进步奖二等奖。

陈　戈

硕士，毕业于华南理工大学，中国通信标准化协会TC1 WG2副组长，从事多年CDN、IPTV、数据通信等领域研究工作，具有丰富的现网研究、开发经验。

陈步华

硕士，毕业于西安电子科技大学和英国诺丁汉大学，专注于CDN、IPTV和大视频应用方面的研究与标准化工作，并参与中国电信全国CDN网络建设规划和测试。

余　媛

硕士，毕业于北京邮电大学，主要研究方向是下一代网络，从事CDN、数据通信等领域技术研究工作。

内容分发网络原理与实践

唐宏　陈戈　陈步华　余媛◎编著

Principle and Practice of CDN

人民邮电出版社
北京

图书在版编目（CIP）数据

内容分发网络原理与实践 / 唐宏等编著. -- 北京 : 人民邮电出版社, 2018.7
ISBN 978-7-115-48803-9

Ⅰ. ①内… Ⅱ. ①唐… Ⅲ. ①计算机网络－网络结构 Ⅳ. ①TP393.02

中国版本图书馆CIP数据核字(2018)第137137号

内 容 提 要

本书分三部分系统地分析了 CDN 基本原理与关键技术、如何选择合适的 CDN 提升业务质量、如何利用开源软件设计自有 CDN 等相关技术及方案。通过本书读者可以较为深入地了解 CDN 关键技术、CDN 市场与业务、CDN 的初步设计与开发。本书主要面向互联网技术人员、电信运营商业务与技术人员、高等院校相关专业教师与学生。

◆ 编　　著　唐　宏　陈　戈　陈步华　余　媛
责任编辑　李彩珊
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：700×1000　1/16
印张：14　　2018 年 7 月第 1 版
字数：297 千字　　2018 年 7 月河北第 1 次印刷

定价：79.00 元

读者服务热线：(010)81055488　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

前言

内容分发网络（Content Delivery Network，CDN）位于应用层之下、传输层之上的中间层，将上层业务所需要的内容通过 CDN 传送到靠近终端用户的位置，可以提升业务的服务质量。

CDN 不是一种新的网络架构，它所使用的技术也是许多成熟技术的组合，如流媒体技术、DNS 技术、HTTP 技术、IP 技术、缓存技术、IT 硬件技术等，但 CDN 的原理却不能简单地认为是这些技术的堆砌，必须把这些技术有机地结合起来，放在一些特定的业务场景下系统地分析，才能较为深入全面地了解 CDN 的关键技术与原理。

CDN 发展初期，其承载的业务单一，带宽要求不高，但近年来由于互联网与电信运营商的业务快速发展，CDN 所承载内容的内容格式、媒体协议、网络协议等都在快速变化，因此对 CDN 提出了许多新的需求，要求 CDN 能够具有兼容多种业务处理、跨网络、面向多种终端等特性，因此了解 CDN 原理也需要了解目前 IP、IT 的新技术。

CDN 除了被认为是一种网络架构、一种内容分发平台外，许多互联网公司还认为 CDN 是一种业务，它们可以向 CDN 服务提供商租用 CDN 服务，也可以通过自建 CDN 平台来为自有业务提供服务，因此作为互联网公司的技术人员，也需要了解租用与自建 CDN 的对比。

本书分三部分系统地分析了 CDN 基本原理与关键技术、如何选择合适的 CDN 提升业务质量、如何利用开源软件设计自有 CDN 等相关技术及方案。

本书的作者均是中国电信具有多年 CDN 研发经验的技术人员，对 CDN 的研发、测试、

运营都有较为深刻的认识，在 CDN 领域取得了许多专利、标准等成果。唐宏负责本书的总体架构设计、关键技术、新技术部分的编写；陈戈负责 CDN 租用、开源 CDN 架构设计部分的编写；陈步华负责 CDN 产业、CDN 选择、CDN 组网部分的编写；余媛负责开源 CDN 部分的编写。另外，梁洁、庄一嵘对本书内容进行了校对、检查并提出了许多建议，黄宇民对书中的部分技术细节进行了修订，卢琳、何道琼对本书的插图与文本进行了编辑，对此深表谢意。

编　者

2018 年 3 月

目 录

第一部分 CDN 基本原理与关键技术

第二部分 CDN 的选择

第三部分　基于开源的自建 CDN 设计

第一部分

CDN基本原理与关键技术

第 1 章

影响互联网应用质量的关键

1.1 互联网应用发展

互联网已经高速发展了 30 多年，从各方面改变了人类的生活，是人类最伟大的发明之一。最初，互联网只是单纯用来进行数据通信的，直到 20 世纪 80 年代末，万维网（即 WWW）出现，使互联网的架构格局发生了巨大的变化，并使其成为了在全球范围内能够实现资源共享和数据传输的分布式网络。

最开始，互联网应用仅仅是一些利用 HTML 等工具制作的网站内容集合。然而，随着互联网的发展，除了网站以外，还产生了电子商务、社交网络、游戏、定位导航等各种各样的互联网应用，但网站业务依然是互联网应用的最主要应用。网站业务作为人们获取新闻要点和第一手资讯的主要来源，已从传统单一的文字形式，飞速发展到音频、视频以及弹幕、增强现实（Augmented Reality，AR）等实时互动、参与感更加真实强烈的信息传递方式。

近年来，除了网站业务以外，网络游戏、流媒体视频、电子商务、社交网络、地图导航等业务也飞速发展，尤其是在移动互联网技术的大规模部署下，各种互联网应用已呈现爆发式的增长，互联网内容的组织也越来越复杂，如图 1-1 所示。互联网内容都是由网页、音频、视频等构成的，这些内容的发展对互联网应用质量有着很大影响。

图 1-1　互联网内容的爆炸式增长示意

互联网应用的内容数量和类型不断快速增长，从最初的页面只包含 2~3 个对象到现在包含众多对象，这就意味着服务器的往返交互次数会随着内容的复杂程度而快速增加，应用加载所需要的时间也会越长。

1.2　互联网应用质量

1.2.1　质量是互联网应用的生命

（1）用户对互联网应用质量的期望

一项研究用户互联网行为的实验结果显示，用户在 4 s 以内看到网站的页面加载完成或者开始产生页面渲染，会让用户产生非常满意的访问体验，这就是互联网行业里著名的“4 s 法则”。当用户向网站发起访问请求时，若超过 4 s，网页还没有被加载下来，30%的用户会选择不再等待，并关掉页面。对于用户来说，6~8 s 的页面加载时间已经是能够忍耐的极限了，因此，8 s 成为了一个临界值。如果网站的加载速度总是超出 8 s，用户会觉得网站的访问体验很

差。假如用户对这个网页有必不可少的访问需求，最多还会等待到 12 s 左右，要是网页依然没有被加载完成，该网站会产生大量用户的流失。上述实验结果见表 1-1，由于用户倾向性地认为网站打开速度越快，其服务质量就越高，因此，用户对网站的响应速度有着很高的要求。

表 1-1　用户访问体验与响应时间的关系

响应时间	用户感觉
0~4 s	很快
4~6 s	有一点点慢
6~8 s	机器在工作
8~12 s	先干点别的
>12 s	不能用了

当用户向网站发起访问请求，而且在很长一段时间内没有获得响应时，用户的忍受程度会随着网站打开时间的变长而降低，甚至会质疑网站本身的可靠性和质量，认为网站服务器存在一些故障。并且，网站缓慢的打开速度可能会让用户忘记原本访问网站的目的，不得不重新回忆下一步要做什么，从而进一步导致用户的访问体验恶化。

速度和性能是两个相对的概念。由于商业模式、应用场景以及功能复杂性等各不相同，每个网站业务都要满足其特定的用户需求。从图 1-2 用户访问电商网站感觉缓慢的后果中可以看出，所有网站都必须在一定时间内对用户做出响应。因此，从一定程度上看，网站速度是用户满意程度和网站性能的关键因素。

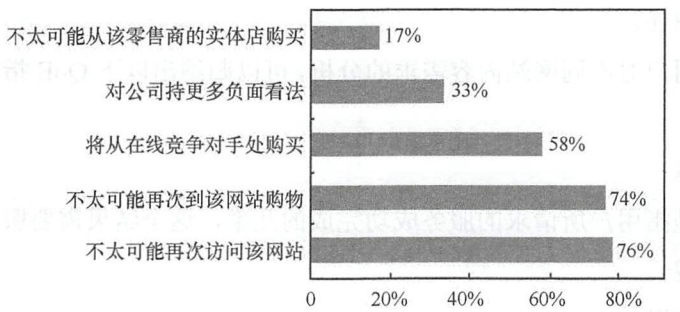


图 1-2　用户访问电商网站感觉缓慢的后果

（2）互联网应用质量指标——QoE

互联网应用的作用是为不同用户提供各式各样的互联网服务，并为用户的生活带来方便，因此，用户的访问体验是至关重要的。然而，每个用户有着主观对服务质量的评价与认识，这是根据用户自身的具体感受而形成的，称为用户体验质量（Quality of Experience，QoE）。此时，可以将 QoE 作为评估互联网应用质量的准则。

QoE 不仅是用户对网站提供的业务质量的主观感受，也包括用户综合设备、网络、系统以及业务可靠性等方面的主观评分。用户通过 QoE 指标对互联网业务质量和性能进行综合评定，促进互联网应用开发者不断优化。

当打开一个普通网页或者一个超文本链接时，用户最希望的是能够成功地立即打开网页。在这个过程中，用户最直接感知到的是页面访问成功率指标。其次，用户能够看到浏览器中是否有响应，称为首个分组时延。此外，从用户请求访问页面开始到页面完全加载所消耗的等待时间被称作页面打开时延。

在网页上播放音频或者视频时，能够流畅观看并且少缓冲是用户对于这类网站业务的需求。在视频加载过程中，用户对等待时间的要求是十分苛刻的。在播放时，用户最不愿意看到的是播放时延，播放时延是由下载速率和播放器的缓冲大小决定的。其次，播放卡顿（即流媒体开始播放后出现等待缓冲的情况）也会对用户体验造成一定影响，其中，下载速率、播放器缓冲大小以及流媒体的源码率等都是带来卡顿的重要因素。此外，播放过程中的花屏现象也是视频加载可能会带来的问题，而用户希望播放的视频是清晰的。

在社交网络或者即时通信业务中，消息能够正常发送并接收是用户的基本诉求。消息发送、接收成功率是用户可以直接感知到的。并且，“即时通信”意味着用户不希望出现通信时延。对于即时通信业务来说，发送并接收的响应速率尤为重要，响应速度越快，就越容易获得更多用户的信任。

通过对普通用户对不同网站内容需求的分析，可以归纳出以下 QoE 指标是用户比较容易感知到的。

- 服务成功率

服务成功率是指用户所请求的服务成功完成的几率，这个结果需要服务建立、数据传输等一系列过程实现。

- 服务建立时间

服务建立时间是指从服务请求到服务呈现所花费的时间，并且会因为用户请求服务内容

的不同而表现出微秒到秒级的区别。

• 时延

时延是一个时间概念，代表用户从发出请求到获得结果的时间，如视频播放时延指用户开始播放到终端开始显示视频图像的时间。

• 视听播放卡顿

视听播放卡顿是指播放音频或者视频时，由于没有接收到足够的数据或较低的帧传输速率而引起的播放暂时停顿的现象。

• 图像清晰度

图像清晰度指的是图片或者视频画面上的细节信息和边界信息的清晰程度以及精确还原实物色彩的能力。

通过介绍用户感知的 QoE 指标，进一步说明了 QoE 是用户在与应用服务交互的过程中产生的主观感受，是对应用质量的判断与期望。

1.2.2　网络性能是影响质量的关键

（1）网络性能对互联网应用质量的影响

网站服务器接入互联网的链路所能提供的带宽是网络流量向用户传送的第一个出口，称为“第一千米”。网站的访问速率和并发访问量是由这个带宽决定的。在用户并发请求和访问量很大的情况下，会产生严重的网络堵塞，并对用户造成不良的访问体验，从而使网站的用户流失。

互联网传输中还容易因为跨电信运营商网络而造成网络拥塞。因为所有电信运营商的网络流量都有着很高的收敛比，并且占比较小。因此，这种情况容易造成互联网的传输拥塞。

再者就是骨干网的拥塞问题。骨干网承载了互联网的绝大部分流量，但是骨干网的流量传输承载能力与互联网应用的发展并不是同步的。通常，当互联网的应用大规模使用和更新时，骨干网的升级和扩容工作还处于缓慢进行的状态中。因此，渐渐地，骨干网的网络流量承载能力就成为了阻碍互联网性能提升的瓶颈。

宽带网络由不同国家、不同地区的电信运营商共同建设的网络互联而成，每个网络往往由骨干网、城域网、接入网等组成，这些不同网络间的时延共同形成了影响互联网访问速率的关键因素。在互联网应用中，从用户发出指令到服务器响应，通常包括以下一系列时延：

终端处理指令时延→网络时延→服务器响应时延→网络时延→终端处理响应时延。

将上述一系列时延过程细化，具体时延指标如下所示。

• 网络时延

网络时延指的是数据分组穿越一个或多个网段所经历的时间。其中，路由处理、ADU（用户数据单元）的传输、服务器对用户请求的处理以及距离产生的网络传输时延，构成了网络时延的主要因素。

• 发送时延

发送时延是指在发送数据时，数据从发送端进入传输介质所消耗的时间，如式（1-1）所示：

$$发送时延=数据帧长度/发送速率 \tag{1-1}$$

• 传播时延

传播时延是指电磁波在传播过程中消耗的时间。数据在特定介质中传输，传播速率是一个常量，如式（1-2）所示：

$$传播时延=传播距离/传播速率 \tag{1-2}$$

• 处理时延

当服务器或路由器接收到数据分组时，会检测数据分组的头部，决定将该数据分组传输到哪一个链路上，这个过程所需的时间被称为节点处理时延。其中，路由器的优劣对处理时延起着决定性的作用。

• 排队时延

排队时延作为 IP 网络的主要时延因素，指的是 PDU（协议数据单元）在传输链路上每一次排队等待所引起的时间延迟的集合。对于 FIFO（先进先出）的队列交换机制，每一个新到达的数据单元都要排队等待前面的数据单元输出完毕。因此，前面的数据单元的数量、队列形式以及输出端口的传输速率都成为了影响排队时延的重要因素。

此外，对于互联网应用提供商而言，除尽可能提高服务器处理效率外，还需在网络部署上考虑降低网络时延。

网络时延除了对用户响应时间造成重要影响外，还对应用的加载速率有巨大影响，以 4K 视频应用为例。目前 4K 视频正在快速应用，8K 也开始进入试商用阶段，4K/8K 视频对带宽、分组丢失等网络质量要求都非常高。DSL 论坛 TR-126 报告指出 4K/8K 等高带宽视频应用所

需的网络质量要求见表 1-2。

表 1-2　高宽带视频对网络质量的要求

视频码率/（Mbit·s^{-1}）	损耗/（错误数·h^{-1}）	分组丢失率
1.5（SD）	1	≤1.6×10^{-4}
5（HD, 1080 P30）	1	≤1.2×10^{-5}
15（UHD, 4K P24/30）	0.25	≤3.9×10^{-6}
30（UHD, 4K P60）	0.25	≤2.0×10^{-6}
60（UHD, 4K P150）	0.25	≤9.8×10^{-7}
100（UHD, 8K P60）	0.25	≤5.8×10^{-7}

从表 1-2 可以看出，4KP60（每秒 60 帧的 4K 视频，也称为真 4K）视频，其每秒所需带宽达 30 Mbit/s。从技术参数来说，在现在光纤入户的情况下，100 Mbit/s、200 Mbit/s 甚至 1 000 Mbit/s 接入带宽已是常见的，观看 4K 内容没有问题，但实际情况并不是这样。

互联网 4K 内容一般通过 TCP 来承载，而 TCP 的最大吞吐速率与网络时延密切相关，如式（1-3）所示：

$$\text{TCP 吞吐量} \leqslant \min\left(\text{BW}, \frac{\text{WindowSize}}{\text{RTT}}, \frac{\text{MSS}}{\text{RTT}} \cdot \frac{1}{\sqrt{p}}\right) \tag{1-3}$$

其中，RTT 表示往返时延，WindowSize 表示窗口大小，p 表示分组丢失率，BW 表示最大带宽，MSS 表示最大分片大小。

由式（1-3）可以看出，如果 RTT 时间与分组丢失率未能达到要求，即使再大的接入带宽也难以达到 4K 接入要求，所以，时延对应用的影响是十分巨大的。

（2）网络传送性能指标——QoS

对于用户能感知到的互联网应用性能的指标，与网络设备与服务性能有着密切的关系。比如，在网络通话中用户听不清楚以及在播放视频中，图像闪动不清晰等，这些往往都是在打开一个网站的响应过程中，传输的时延、数据丢失以及带宽容量限制所造成的。

而这些造成不良用户体验的因素——网络的时延、抖动、带宽以及分组丢失统称为服务性能（Quality of Service，QoS）指标。可以说，QoS 的性能要求源于用户的需求——用户体验质量（QoE）。通过对网络设备层进行监控和分析，QoS 客观地反映了网络的服务性能。具

体来说，QoS 可以被理解为底层分组数据传输的性能指标。

• 时延

从用户的角度来说，时延就是从用户发出请求到接收到远端服务器做出响应所花费的时间。基于 TCP/IP 的网络传输包括路由器处理、用户数据单元传输以及服务器处理这一系列过程，从而将产生相应的路由时延和传输时延等。

• 抖动

抖动也被称作时延的变化，即时延的增量，是各种时延的变化导致网络中的数据分组到达速率的变化。数据在网络传输中要通过许多链路，由于各个数据分组在传输中经受的时延不同，因此，破坏了原来的数据分组顺序，产生了抖动现象。

• 分组丢失

在当前高速网络中，数据分组丢失是影响网络性能的关键因素之一。分组丢失率较低时，会出现视频播放花屏现象，通话质量不清晰，打开网页时快时慢；严重分组丢失时会出现速度过慢、网络连接断开等情况，导致网络不可用。分组丢失是路由队列满负荷时，没有多余的空间再去承载其他数据分组而引起的。分组丢失率是指丢失的数据分组与所有数据分组的比值，通常在 0~15%变化，而 15%已经是极为严重的分组丢失情况了。

• 带宽

带宽通常指链路中的端到端传输速率，如平常所说的 100 Mbit/s、1 000 Mbit/s 等，带宽与物理链路、时钟速率、接口等有着重要的关联，可以用来表示网络的状况。

从本质上说，QoE 以用户为中心，更加关注用户的使用体验。而 QoS 更加关注于服务层面，目的是实现以服务为中心的管理。QoS 的关键在于从网络服务性能的角度来处理业务，实现为用户提供满意的服务质量。

1.3 提高互联网应用质量的方法

1.3.1 集中式部署带来的问题

在前面介绍过，对于用户来说，网站下载速率的可靠性带来的良好体验是为网站带来越

来越多的用户的重要因素。但随着用户对互联网内容的多样化需求增长，网络资源不断丰富，使得互联网服务内容产生了巨大的变化，其中，视频业务流量的骤增给基础网络带来极大的冲击，并且会经常产生网络异常流量。

这些异常突发流量造成的时延、分组丢失、抖动等因素，给网络带宽造成了巨大压力，大大影响了用户的访问体验。这是由于早期的互联网业务往往都集中部署在一个或若干个大的节点上，但这种集中式部署的系统具有明显的单点问题。尽管大型服务器通常在稳定性和可靠性上展现了卓越的性能，然而再“优秀”的服务器仍然会发生故障，从而影响服务性能。万一单点故障出现，则整个服务系统将处于无法使用的状态，造成极为严重的后果。另外，互联网应用的不断发展而使得用户访问量大大提高，整个服务系统的规模也在不断扩大，因此，在单一节点上完成系统的扩容是十分困难的。另一方面，集中式部署也会带来网络时延过大的问题。

因此，现在大型服务器的市场占有率变得越来越小，大多数企业也开始放弃采用传统的大型服务器集中式部署方案，而纷纷选择利用小型服务器来搭建分布式服务器系统，实现就近服务，提升用户体验质量，如图 1-3 所示。

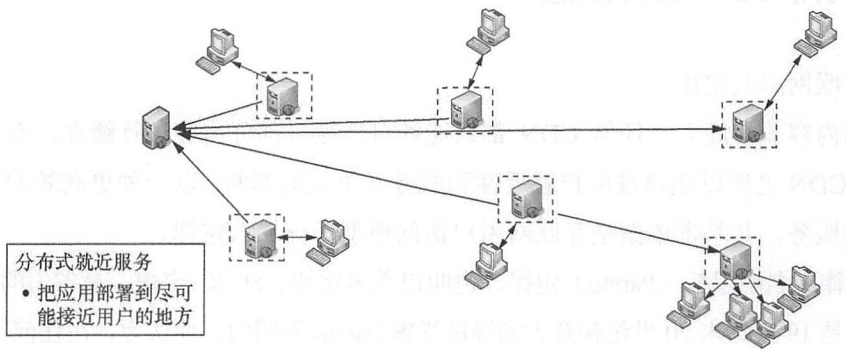


图 1-3　分布式就近服务

1.3.2　利用网站镜像加速

让用户快速获取需要的网站资源并提高网站的服务质量已经成为改善网络应用环境的主要目标。最初，网站建设者使用镜像技术来解决服务器时延、单点部署以及“第一千米”的难题。

网站镜像是通过复制整个网站或部分网页内容并将其重新存储到其他服务器，它让用户在这些服务器依然能够访问想获取的内容，并且为主站分担了网络流量，称为主站的后备资源。将网站或者网站的一部分按原来的内容、结构以及功能完全复制，就形成了一份镜像。镜像和主站的区别只是体现在处于不同的服务器，而呈现的内容却是一模一样的。网站镜像通常适用于静态内容同步。

网站镜像通常在主站流量过高，服务器无法承受时，为主站服务器分流减压。这项优势在主站服务器宕机的情况下表现得尤为明显，由于网站镜像的存在，用户请求仍然能够通过其他服务器正常响应。在不同电信运营商网络中，往往会分别部署网站镜像，因此，相对于不同电信运营商的用户来说，利用网站镜像能够加快用户的访问速率。

然而，通常都是个人网站的服务器因为无法承受太多的用户访问量而采取网站镜像的策略，商业网站一般并不采用镜像的办法。由于内容需要重复上传，镜像和主站随时能够保持一致，这样在需要重复上传的内容过多时会显得非常麻烦。因此，目前网站镜像已经应用得越来越少，逐渐被 CDN 所取代。

1.3.3 利用 CDN 进行加速

（1）互联网二八定律

互联网内容非常庞大，任何 CDN 都只能针对性地对部分内容进行缓存，而无法缓存所有的内容。CDN 之所以能通过用户就近性和服务器负载的判断，以一种更高效的方式为用户的请求提供服务，其基本依据是互联网用户访问模型——二八定律。

二八定律又名帕累托（Pareto）定律，也叫巴莱多定律、80/20 定律、最省力的法则、不平衡原则等，是 19 世纪末 20 世纪初意大利经济学家 Pareto 发明的。他认为，在任何一组东西中，最重要的只占其中一小部分，约 20%，其余 80%尽管是多数，却是次要的，因此又称二八法则。

二八定律也同样适用于互联网用户访问，即大部分用户都只是访问互联网的少量热点内容，其他大部分内容访问的人是很少的。因此，CDN 可以通过自身的算法，只缓存热点内容就可以为大部分用户提供服务。

通过对互联网用户访问行为的进一步分析，人们发现，内容访问近似符合齐普夫（Zipf）定律。这个定律是美国语言学家 Zipf 发现的，他在 1932 年研究英文单词的出现频率时，发现如果把单词频率按照从高到低的次序排列，每个单词的出现频率和它的符号访问排名存在

简单反比关系，如式（1-4）所示：

$$P(r)=\frac{C}{r^a} \tag{1-4}$$

这里 r 表示一个单词的出现频率的排名，$P(r)$表示排名为 r 的单词的出现频率。其中，在单词频率分布中 C 约等于 0.1，a 约等于 1。

该分布称为 Zipf 分布，是一个统计型的经验规律，描述了这样一个定理：只有少数英文单词经常被使用，大部分的单词很少被使用。这个定理也在很多分布里面得到了验证，比如人们的收入、互联网的网站数量和访问比例、互联网内容和访问比例，a 越大，分布越密集，对于视频点播访问情况来说某些时候符合双 Zipf 分布。

Zipf 分布可以看成二八定律的数学抽象函数。根据 Zipf 分布的参数调节，可以对 CDN 模型进行各种优化研究。

下面是某个视频系统内容的访问分布，图 1-4 所示为热度统计曲线。可以看到，多数访问集中于少量内容上。

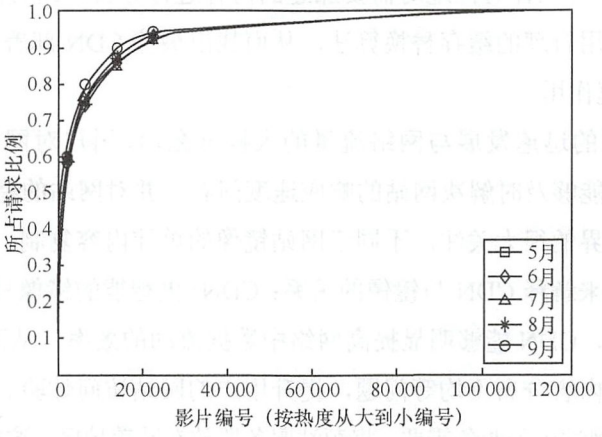


图 1-4　热度统计曲线

在连续跟踪多个月份后，可以看到排名前 2 万的影片，其点播的总次数占所有 12 万影片的总次数 90%以上。这意味着，CDN 的服务节点只需缓存 2 万部影片即可满足全部用户的 90%以上的访问需求。

图 1-5 所示为对数轴的访问频率曲线，可以看到近似为一条直线，符合 Zipf 分布的对数坐标表示。

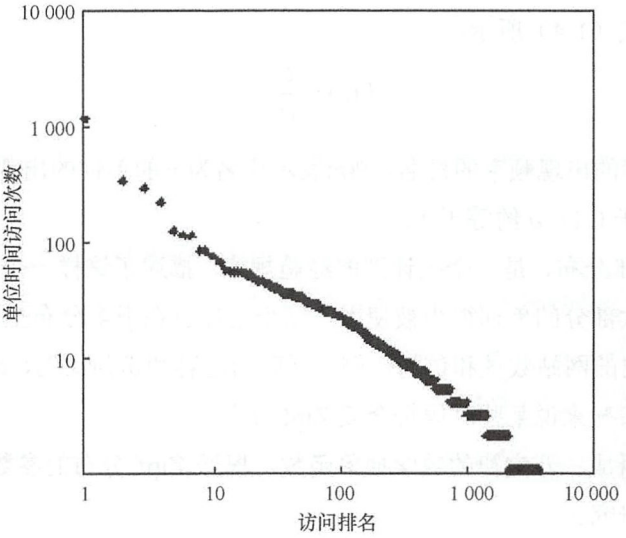


图 1-5　访问频率曲线

根据 Zipf 分布，可以针对性地对需要加速的内容进行分析，结合网络条件，设计 CDN 各级存储的比例，采用合理的缓存替换算法，从而找出最佳 CDN 部署方案。

（2）CDN 的加速作用

随着互联网应用的迅速发展与网络流量的大幅度激增，用户对网站的加速需求日益增长。由于 CDN 技术能够及时解决网站的响应速度问题，并对网站的稳定性起了较大的提升作用，因此受到了业界的很大关注。不同于网站镜像的单纯内容复制，CDN 技术更加智能，可以用这样一个式子来解释 CDN 与镜像的关系：CDN=更智能的镜像+缓存+流量调度。从上面的关系式可以看出，CDN 能够明显提高网络中数据流动的效率，从而解决网络带宽不足、用户访问量过大以及内容分布不均等问题，提升用户的网站访问体验。

许多我国国内的网站出于业务需要，将源站服务器放在欧美地区。这样一来，物理距离距中国太远，普遍 Ping 所需的时间都在 100 ms 以上，使网站的用户会感觉到访问速率比较慢，访问体验度方面有所下降。所以 CDN 技术首先要解决的就是物理距离远所导致的访问速率降低问题。通过 CDN 技术，在中国香港、中国台湾等地区和日本、韩国等国家部署 CDN 节点进行数据分发，即使源站放置在遥远的欧美地区，中国用户的访问速率也会得到明显的改善。

最初 CDN 的提出，就是为了通过就近提供服务来解决物理距离过远导致性能不好的问题。使用 CDN 后，网络的基本组织架构和内容传输情况发生了很大变化。从普通网站用户

的角度上看，CDN 节点的作用就相当于把一个网站就近部署在用户周围。CDN 服务器会像源站服务器一样，为用户提供需要的内容服务。但是，由于 CDN 节点更靠近用户，因而能够更快地响应用户的请求。以视频网站为例，图 1-6 是 CDN 为视频网站进行加速的示意图，使用 CDN 服务后，对服务请求进行了优化调度，更加有效地利用了带宽资源，使得视频加载时间减少，性能提高，如图 1-7 所示。

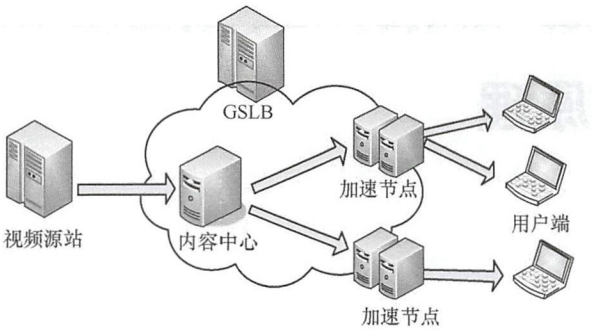


图 1-6　CDN 为视频网站进行加速的示意

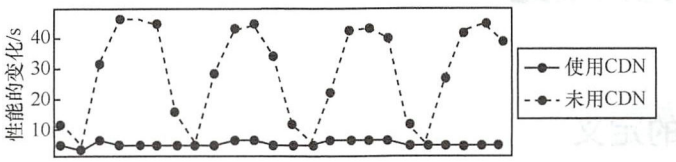


图 1-7　某视频网站使用 CDN 前后性能对比

总的来说，CDN 对互联网应用的优化作用主要体现在以下几个方面：

- 缓解源站服务器访问压力，解决服务器端的“第一千米”问题；
- 优化热点内容的分布，合理缓存，减轻骨干网传输的流量压力；
- 提升用户的访问质量和体验，全面提高网站访问速度；
- 增强网站服务的可靠性，解决网站突发峰值流量问题；
- 解决不同电信运营商之间互联互通问题造成的影响；
- 提高安全性，有效防止异常流量对源站的攻击。

第2章

CDN基本原理

2.1 CDN的基本概念

2.1.1 CDN的定义

内容分发网络（Content Delivery Network，CDN），是在现有网络中增加一层新的网络架构。CDN将源站的内容发布和传送到最靠近用户的边缘地区，使用户可以就近访问想要的内容，从而提高用户访问的响应速度。

CDN的基本原理是依靠放置在各地的缓存服务器，通过全局调度以及内容分发等功能模块，将用户需要的那部分内容部署到最贴近用户的地方，将原本低效、不可靠的IP网络转变成高效、可靠的智能网络，满足用户对内容访问质量的更高要求，改善互联网网络拥塞问题，提高用户访问网站的响应速度。

从字面意义上可以看出，CDN 的构成元素为内容（Content）、分发（Delivery）以及网络（Network）。

（1）内容

CDN的内容通常是以下两种：静态内容以及动态内容。

静态内容：内容不经常更改，并且一旦它在 CDN 缓存中，可以由许多用户进行访问，缓存性强。

动态内容：内容用于特定的用户或组，并且更新频率较高，通常来自源服务器并实时发送到 CDN 中，缓存性较弱。对于用户的每一次请求，CDN 都必须从源站服务器拉取动态内容，所以动态内容加速的常用方法就是降低源站服务器和用户终端之间的传输时延。

（2）分发

CDN 的分发是指利用一定传送策略，将用户请求的内容发布到距离该用户最近的节点。

（3）网络

CDN 由成千上万个分布式服务器组成，通过服务器的通信，把内容分发和传送给终端用户。CDN 各节点之间是通过电信运营商的宽带网络进行通信的，可以说 CDN 是在电信运营商的网络之上的一层网络。

2.1.2　CDN 可承载的内容

在第 1 章介绍过，用户在向网站发起访问请求时，如果等待一定时间网站还没有响应，用户就会放弃访问，而镜像通常不适用于大规模商业网站加速，因此，CDN 加速需求应运而生。

静态内容是最早出现的 CDN 承载的内容类型，以文字、图片、动画等更新频率低的内容为主，因此，CDN 技术最初就是用来对这些静态内容网页进行加速的。一个典型的静态内容页面如图 2-1 所示。

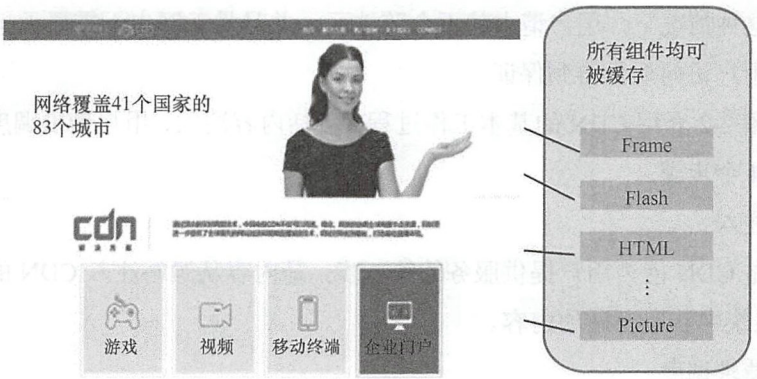


图 2-1　典型的静态页面

后来，随着互联网的大幅度升温、宽带的普及，用户利用互联网下载所需文件已经成为一种习惯，因此，CDN 对下载业务的加速服务也是必不可少的。

近年来，大量视频网站涌现，流媒体流量随之迅速攀升，从而驱动了 CDN 技术的应用重点也逐步转为流媒体加速服务。

随着互联网技术的发展，社交网络、在线支付以及网络游戏等实时性强、内容经常更新的互联网应用逐渐产生，因此，CDN 技术也从静态内容的加速发展到动态内容的加速。

因此，从互联网应用的角度看，需要 CDN 承载的内容主要为静态内容和动态内容。将在第 5 章详细介绍 CDN 对这些承载内容的加速技术。

2.2 CDN 的工作过程

2.2.1 CDN 的基本工作过程

CDN 服务与传统网络服务最大的差别在于访问方式。传统情况下，用户发起访问请求后，对于同一个内容的所有用户请求，都集中在同一个目标服务器上。

而利用 CDN 加速后，用户的内容请求解析权交给了 CDN 的调度系统，然后将用户请求引导到性能最佳的最靠近用户的 CDN 节点上，最终该节点为用户请求提供服务。

传统的访问方式，造成了在网络中传输的极大压力，并且还无法保证用户的良好访问体验。而使用 CDN 服务后，用户的访问请求不会集中在相同的目标服务器上，而是会分散到不同节点，在这种情况下，用户请求就不会跨地区，并且骨干网也不需要承担过重的流量负担，进而使得用户访问质量得到保证。

下面按照图 2-2 介绍 CDN 的基本工作过程，包括内容注入、用户请求调度、内容分发以及内容服务这 4 个步骤。

（1）内容注入

内容注入是 CDN 能为用户提供服务的第一步，是内容从源站注入 CDN 的过程，使得用户能从 CDN 系统中获取源站的内容。

（2）用户请求调度

用户请求调度是用户向网站发起访问请求，最终用户被引导到最佳的有内容的 CDN 节

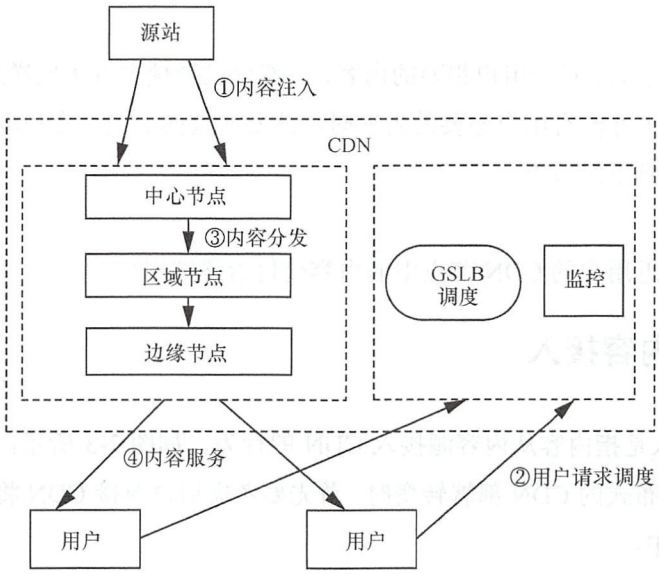


图 2-2　CDN 工作的基本过程

点的过程，具体如下：

（a）当用户向网站发起访问请求时，经由本地 DNS 系统解析，本地 DNS 会通过递归方式将域名的解析权最终交给 CDN 授权 DNS 服务器（GSLB）；

（b）CDNGSLB 可将 CDN 节点设备的 IP 地址返回用户，也可以将另一个负责解析用户终端 IP 地址的 GSLB 设备的 IP 地址返回用户（细节参见第 4.3 节）；

（c）用户向 CDN 的 GSLB 设备发起内容访问请求（IP 调度方式）；

（d）CDN 的 GSLB 设备根据用户 IP 地址以及用户请求的内容 URL，选择一台用户所属地区的本地负载均衡（SLB）设备，并让用户向该 SLB 发起访问请求；

（e）该 SLB 设备通过决策选择一台最佳的服务器为用户提供服务，用户向该服务器发起访问请求；

（f）若该服务器内容未命中，而 SLB 仍将该服务器分配给用户，则该服务器需要向上级节点请求内容，然后，由该服务器向用户提供“边拉边放”的服务或者由上级节点直接为用户提供服务。

在本书的第 4 章，会详细介绍用户请求全局负载均衡（GSLB）调度与本地负载均衡（SLB）技术。

（3）内容分发

当用户发起请求时，对于用户想要的内容，一部分已经预先直接推送到靠近用户的节点；但是，当下级节点上并没有用户想要的内容时，就要通过向上级节点拉取内容的方式，把用户想要的内容分发到下级节点。

（4）内容服务

把找到的最靠近用户的 CDN 节点中的内容交付给终端用户。

2.2.2 CDN 内容接入

CDN 内容接入是指内容从内容源接入 CDN 的行为，如图 2-3 所示。当互联网应用希望由集中式部署向分布式的 CDN 部署转变时，首先要考虑通过对接 CDN 将现有集中式部署的内容转移到 CDN 中。

CDN 内容接入有 3 种接入方式：内容存储接入方式、内容预注入方式、实时回源方式，这 3 种内容接入方式的适用场景及业务流程均有较大不同。

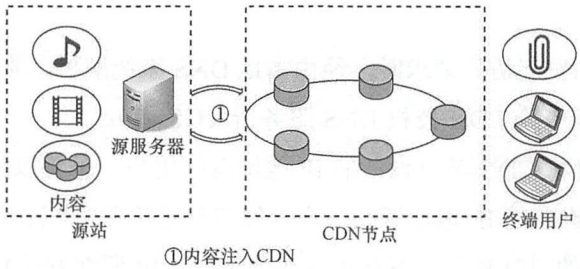


图 2-3　内容注入示意

（1）内容存储接入

内容存储接入指源站（互联网应用的内容源）在发布内容前，提前把内容注入 CDN。内容存储接入方式下，业务系统需主动向 CDN 内容库发送操作指令，CDN 根据指令获得内容并存储在 CDN 内容库中，从而在终端访问 CDN 时直接由 CDN 向终端提供内容，无需再从源站获取，提升了终端用户体验。

采用内容存储接入方式接入的内容将永久存储在 CDN 中，直到通过内容接入操作指令对该内容显式删除。CDN 的内容存储接入包括对注入内容的增加、删除和更新，能够通过业务系统或手工方式主动发起内容删除操作并立即实现全网删除。

（2）内容预注入

内容预注入是指源站在内容发布之前将内容注入 CDN 中。内容预注入与内容存储接入方式类似，都是由业务系统主动向 CDN 发送操作指令，CDN 根据指令预先从内容源回源获取内容，是就近提供服务的接入方式。

但采用内容预注入方式接入的内容并不永久存储在 CDN 中，而仅仅是进行内容缓存，CDN 会根据内容访问的热度情况对缓存内容进行智能删除，预注入内容可以设定一段时间不被删除的内容保护期。采用内容预注入方式接入的内容当被缓存删除后，CDN 仍可以通过回源方式获取内容提供服务。

（3）实时回源

实时回源（未注入）是指源站在内容发布之前不向 CDN 注入内容，但当用户内容访问请求时，CDN 实时地从源站拉取内容。

内容回源是指对于非托管模式的内容接入，当 CDN 收到业务系统内容预注入指令或用户内容服务请求而本地没有内容时，向内容源请求并获取内容接入 CDN 的行为。实时回源方式无需由业务系统主动向 CDN 预先注入内容，而是在终端访问 CDN 时，通过回源方式向内容源实时获取内容到 CDN 中，向终端提供后续就近缓存服务。

内容存储接入方式对用户的服务质量保障最佳，但对 CDN 的资源消耗较大，成本较高，适用于 IPTV 等对质量要求极高的业务应用。实时回源获取方式对 CDN 资源消耗较小，成本较低，但对用户的服务质量保障比不上内容存储接入方式，一般在网站等业务应用上使用，是目前 CDN 的最主要接入方式。内容预注入方式介于内容存储接入与实时获取方式，互联网服务提供商可根据自有业务的需求选择合适的内容接入方式。

2.2.3　CDN 用户请求调度

通常情况下，CDN 用户的内容访问请求调度如图 2-4 所示，分为两个层次：全局调度和本地调度。

（1）全局调度

全局调度的主要目的是根据用户所在地理位置不同，在各个节点之间进行分析决策，将用户请求转移到整个网络中最靠近用户的节点。全局调度方式目前主要有基于 DNS 调度方式和基于应用层重定向调度两种方式，在第 5 章将详细介绍这两种全局调度方式。

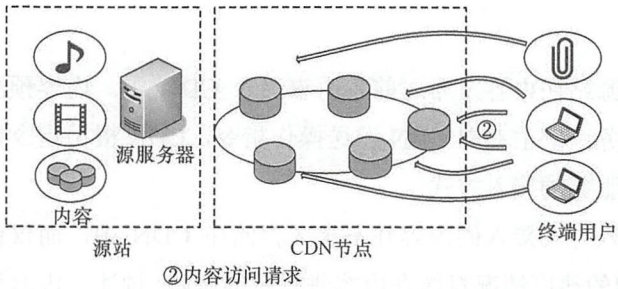


图 2-4　用户请求调度示意

（2）本地调度

和全局调度系统相比，本地调度通常被限制在一定地区范围内，并且更加关注 CDN 服务器设备具体的健康状况与负载情况，根据实时响应时间，将任务分配给最适合的服务器设备进行处理，进行更精细粒度的调度决策，实现真正的智能通信和发挥服务器集群最佳性能。本地调度的意义在于充分利用现有设备，有效地解决了用户访问请求过多引起的系统负载过重的问题。

2.2.4　CDN 内容分发

互联网应用的响应时间通常是由网络带宽、路由时延、网站处理能力以及物理距离等因素决定的。其中，物理距离过长对互联网应用的响应时间有最直接的影响，会使响应速度变得十分缓慢。因此，利用 CDN 技术把最热的内容分发部署到各地的节点上，如图 2-5 所示。内容为不同地区的用户提供就近服务，就能够有效地提高互联网应用的响应速度。

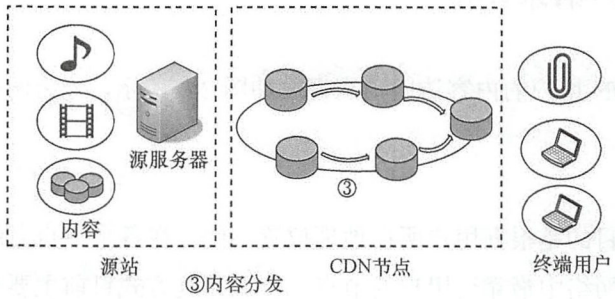


图 2-5　内容分发示意

内容的分发有 Push 和 Pull 以及混合分发共 3 种实现方式。

（1）Push 方式

Push 是一种主动分发的方式。通常，Push 由 CDN 内容管理系统发起，将内容从源站或者中心内容库主动分发到各边缘的 CDN 节点，分发的协议可以采用 HTTP、FTP 等。通过 Push 分发的内容一般是比较热点的内容，这些内容通过 Push 方式预先主动分发到边缘 CDN 节点，可以实现有针对性的内容提供。对于 Push 分发需要考虑的主要问题是分发策略，即在什么时候分发什么内容。

一般来说，Push 分发是一种智能的主动分发策略。可以通过用户访问的统计信息（例如，热度级别排序信息）和已经预先设定的内容分发的规则，智能地决定是否进行内容主动分发。并且可以根据用户历史访问数据等，建立回归模型，对于智能预测用户可能会大量访问的内容，将其提前推送到边缘节点。

（2）Pull 方式

Pull 是一种被动的分发方式，Pull 分发通常由用户请求驱动。当用户请求的内容在本地的边缘 CDN 节点上不存在（未命中）时，该 CDN 节点启动 Pull 方式从内容源或者其他 CDN 节点实时拉取内容，并且在 Pull 方式下，内容是按需分发的。

在实际的 CDN 系统中，一般两种分发方式都支持，但是根据内容的类型和业务模式的不同，在选择主要的内容分发方式时会有所不同。通常，Push 方式适合内容访问比较集中的情况，例如，热点的流媒体视频内容；Pull 方式比较适合内容访问分散的情况。

（3）混合分发方式

混合分发方式就是 Push 与 Pull 分发方式结合的一种机制。混合分发方式有多种方案，最常见的混合分发机制，是利用 Push 方式进行内容预推，后续则使用 Pull 方式拉取。

混合分发方式能够根据当前内容分发系统中的内容服务状况，采用推拉的方式动态地调整内容在内容分发系统中的分布，对于热点内容主动将其推送（缓存）到边缘节点。

2.2.5　CDN 内容服务

CDN 内容服务是 CDN 工作过程的最后一步，内容服务的质量往往决定了业务的质量，也是直接为用户提供服务的关键一步，如图 2-6 所示。CDN 内容服务根据用户访问的内容分类，可分为静态内容服务、动态内容服务等。

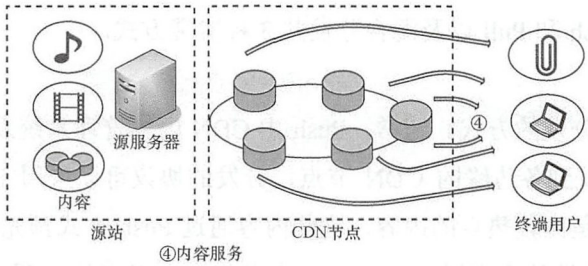


图 2-6　CDN 内容服务示意

（1）静态内容服务

静态内容服务是指 CDN 承载的内容主要是 HTML 文件、文本、Flash 动画、图片、视频、应用软件等。一般来说，这些文件相对更新频率较低，而且热点内容集中，通过缓存技术可将热点内容分发存储在 CDN 的节点上，以满足终端用户就近访问的需求。

静态内容服务又可分为缓存类服务及流媒体类服务。

缓存类服务，即用户以最大的速度向服务器下拉内容，如基于 HTTP 的文件下载，CDN 能够保证下载业务中终端用户的下载速度。

流媒体类服务，主要是视频业务，分为点播与直播两种。与静态内容不同，流式传输方式是流媒体与静态内容最大的差异。流媒体服务将每一帧数据打上时序标签后进行流式传输，是一种按视频的码流需求向用户提供实时流的分发方式。发送端采集音视频数据，经过编码、格式封装、推流等过程处理，然后进行传输，客户端再下载数据并按时序进行解码播放，实现这些离不开 CDN 对流媒体协议的支持。

同时，流媒体服务是互联网中最消耗带宽的传输形式，所以，视频网站对流媒体服务器带宽的需求是非常高的，因此，可以使用 CDN 服务器对视频服务器进行分流减负。

（2）动态内容服务

随着互联网业务的发展，网站内容逐步个性化并且经常更新，如人人网、新浪微博以及微信等社交网络的个性化实时内容。动态内容作为源站动态生成的内容，其对每个用户都不一样，是实时产生的，因此对于动态生成的内容，用户请求需要回源获取内容，CDN 主要通过降低传输时延和避免数据传输失败来实现动态内容加速。

第 3 章

典型的 CDN 架构与组网

不同功能、不同运营商、不同应用场景下的 CDN 架构与组网有很大不同。但这些不同的 CDN 在逻辑功能、主要关键模块划分、部署思路方面都有很多相似之处。通过分析一个典型的视频 CDN 的架构与组网，介绍 CDN 架构设计与组网部署的一般思路。

3.1 CDN 功能平面

CDN 从功能上可以划分为包含管理平面、调度平面和数据平面在内的 3 个逻辑平面。其中，在管理平面的管理和控制下实现内容分发和推送，在调度平面完成用户请求的调度、控制以及各种内容调度策略，在数据平面实现内容分发与服务实体。CDN 逻辑平面的划分与表示如图 3-1 所示。

CDN 管理平面主要用来完成业务管理、网络管理、分发策略管理、内容接入管理以及回源分发管理等一系列管理功能，管理、监控并保障 CDN 承载业务的高效运营。

CDN 调度平面主要实现用户服务请求调度（包括 DNS 调度、HTTP 调度、RTSP 调度）、内容定位、内容路由等功能，通过控制用户服务请求的调度，实现对用户的就近及有保障的服务。

CDN 数据平面主要是内容分发与服务实体，主要负责为用户提供内容分发与应用服务，包括内容存储、内容缓存、内容分发、内容转码、内容服务等功能。

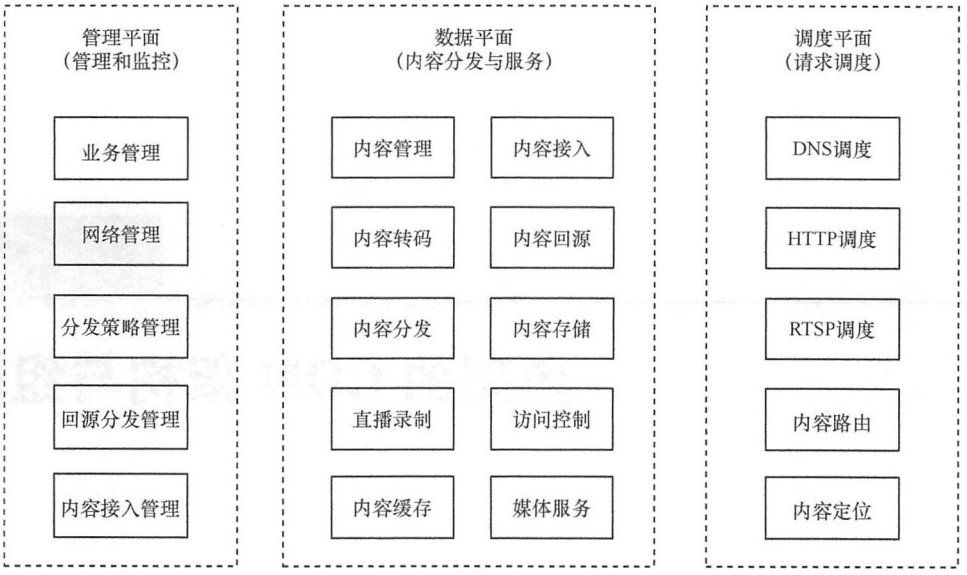


图 3-1　CDN 逻辑平面的划分与表示

3.2　CDN 内部网元

CDN 系统的总体架构及相关服务接口如图 3-2 所示。

CDN 内部逻辑架构主要包含内容中心节点、区域节点、边缘服务节点、全局内容路由 RR、CDN 管理模块以及接口等[1]。

（1）内容中心节点

主要负责和业务系统对接实现 CDN 内容接入、管理、存储和主动分发到 CDN 各媒体服务节点中。具体包含以下功能。

- 内容接入：根据 CMS 的内容注入指令获取指定内容并注入内容存储上，同时在内容管理模块进行登记；或者不通过 CMS 注入内容，而是使用回源方式接入内容源[2]。
- 内容分发：根据内容管理模块的调度策略分发和传送内容。
- 本地负载均衡（SLB）：接收下级节点的内容定位与请求，进行内容寻址，并根据负载均衡策略分配合适的设备提供服务，对节点内设备进行负载均衡[2]。

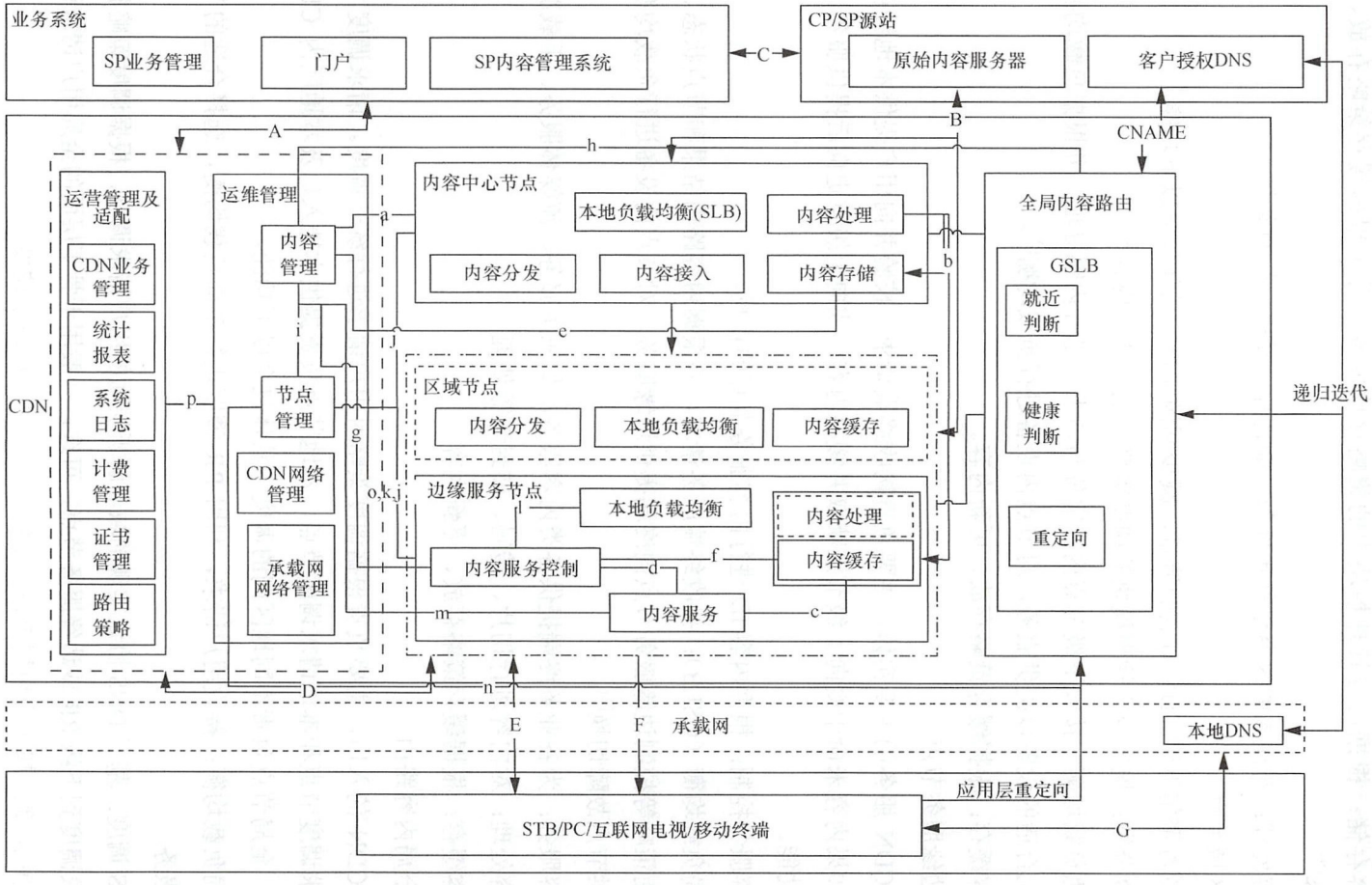


图 3-2　CDN 逻辑架构及相关服务接口

- 内容存储：根据内容管理模块中的策略存放内容，并可根据缓存策略存放、更新内容[2]。
- 内容处理：对注入的内容进行预处理，如切片、转码、转封装等。

（2）区域节点

主要负责聚集下级 CDN 节点的流量，减少回源流量。具体包含以下功能。

- 内容分发：根据内容管理模块的调度策略分发和传送内容[2]。
- 本地负载均衡：接收下级节点内容定位与请求，进行内容寻址，并根据负载均衡策略分配合适的设备以提供服务，对节点内设备进行负载均衡[2]。
- 内容缓存：根据缓存策略存放、更新内容。

（3）边缘服务节点

作为 CDN 服务的主要实体，主要负责接收终端请求，校验并向用户提供本地缓存的内容服务，如果内容未命中则向上级节点获取并缓存内容，或向上级重定向后提供服务。具体包含以下功能。

- 内容服务控制：根据内容 ID 进行内容服务节点的查找。
- 全局负载均衡（GSLB）：接收终端服务请求，并根据内容路由结果和节点状态、负载均衡策略等控制边缘服务节点向终端提供边缘服务，对节点内设备进行负载均衡，并提供节点故障切换[2]。
- 内容服务：为各业务终端提供各类内容服务，如应用服务、流媒体服务、下载服务。
- 内容处理：对内容进行切片、转码、转封装等处理。
- 内容缓存：根据缓存策略存放、更新内容。

（4）全局内容路由

作为 CDN 的入口，主要负责根据调度策略对用户请求进行统一调度。请求调度可区分为全局请求调度节点和本地请求调度节点，本地节点负责提供本地入口及本地区内 CDN 服务的调度，全局节点负责不同地区间的调度。具体包含以下功能。

- 全局负载均衡：接收用户请求，利用 RR 调度进行全局负载均衡，选择合适的节点提供服务。
- DNS 调度：基于 DNS 协议实现的域名解析层面的用户请求调度，根据调度策略将用户请求调度到下级 RR 或边缘服务节点，可选，一般用于骨干层面的全局用户请求调度。
- RTSP 调度：基于应用层 RTSP 重定向实现的用户请求调度。

- HTTP 调度：基于应用层 HTTP 重定向实现的用户请求调度。
- 调度策略管理：定义 CDN 用户请求的调度策略。
- 节点状态管理：查询和管理 CDN 各节点的负载情况，作为调度的基本信息。
- 内容视图管理：管理所辖区域内 CDN 各节点内容的分布情况，可选，当 RR 和 SLB 合设时需要配置。

（5）运维管理

负责对 CDN 的端到端业务质量监控、故障分析定界、内容管理等功能，实现对 CDN 上承载业务的监控和管理，具体包括以下功能。

- 内容管理：负责内容在 CDN 内各项属性的信息登记与管理，包括内容 ID、元数据信息、生命周期、内容状态、分发与推送策略、内容更新策略等功能[3]。
- CDN 网络管理：负责对 CDN 网络性能、告警、拓扑等信息进行管理[3]。
- 业务管理：负责对 CDN 业务配置，如内容注入、CDN 服务、内容路由、调度策略等进行管理[3]。
- 承载网网络管理：对 CDN 的承载网进行管理，包括网络性能、告警、拓扑等信息管理。

（6）运营管理及适配

负责 CDN 内部网络管理、业务管理、报表统计等功能，并可接收适配统一运营管理系统指令转换成 CDN 内部的网络管理操作。具体包含以下功能。

- CDN 业务管理：负责进行 CDN 业务配置，如内容注入、CDN 服务、内容路由、调度策略等进行统一化配置[3]。
- 服务统计：提供 CDN 服务数据统计功能[3]。
- 系统日志：收集 CDN 各节点的服务日志。
- 计费管理：对外服务的 CDN 需对租用方提供计费信息，对原始计费信息根据计费策略进行计费调整。
- 证书管理：针对 HTTPS 服务进行客户证书管理。
- 路由策略：对 CDN 节点上级节点路由进行配置。

（7）接口

CDN 接口根据其对接的系统不同，分为对外接口及内部接口。根据系统和组网架构，CDN 对外接口见表 3-1[3]。

表 3-1　CDN 系统对外接口

接口	定义	协议	备注
外部业务系统与 CDN 管理接口	A	SOAP+XML	该接口功能为管理配置、资源上报、业务管理信息
源站系统与 CDN 接口	B	SOAP+XML、FTP、HTTP	该接口功能为把外部内容及相关元数据引入 CDN 系统
外部业务系统与源站系统间接口	C	SOAP+XML	该接口把相关内容对应的元数据信息引入 CDN 系统
CDN 管理系统与 CDN 内容服务控制功能模块间接口	D	SOAP+XML	该接口负责管理服务状态信息，客户端访问信息
客户端与边缘服务节点控制功能模块间接口	E	RTSP、HTTP	内容交付请求与控制
客户端与边缘服务节点内容服务功能模块间接口	F	RTSP、HTTP	内容的服务与传输
客户端与 DNS、RR、GSLB 间接口	G	DNS、HTTP、RTSP	内容服务定向信息交互、客户端对内容控制请求信息交互

CDN 对内接口见表 3-2。

表 3-2　CDN 系统内部接口

接口	定义	协议	备注
内容管理和内容预处理功能模块之间的接口	a	HTTP	该接口功能为进行内容分发策略的交互
内容预处理与内容存储功能模块之间的接口	b	HTTP	该接口功能为分发、接收及中继元数据和内容信息
内容存储与内容服务功能模块之间的接口	c	RTSP、HTTP	该接口功能为内容服务所需数据的读取
内容服务控制与内容服务功能模块之间的接口	d	HTTP	该接口功能为内容交付指令信息的交互、客户端访问信息交互
内容管理与内容存储功能模块之间的接口	e	HTTP	该接口功能为内容信息的存储状态信息交互，对调度的内容进行存储管理和存储控制等
内容存储与内容服务控制功能模块之间的接口	f	HTTP	该接口功能为内容存储信息交互，以支持内容服务全局/局部重定向功能等

（续表）

接口	定义	协议	备注
内容管理与内容服务控制功能模块之间的接口	g	HTTP	该接口功能为内容服务功能模块内容缓存状态信息
服务路由与内容管理功能模块之间的接口	h	HTTP	该接口功能为内容地址信息交互以支持内容服务定位等
节点管理与内容管理功能模块之间的接口	i	HTTP	该接口功能为节点资源信息，节点网络拓扑信息交互
节点管理与内容存储功能模块之间的接口	j	HTTP	该接口功能为内容存储功能节点注册认证信息以及资源信息交互
节点管理与内容服务功能模块之间的接口	k	HTTP	该接口功能为内容服务节点节点注册认证信息、资源和使用状态信息交互
服务路由与内容服务控制功能模块之间的接口	l	HTTP	该接口功能为客户端对内容服务请求信息和内容服务定位信息的交互
内容管理与内容服务功能模块之间的接口	m	HTTP	该接口功能为内容服务功能缓存相关内容信息的交互
服务路由与节点管理功能模块之间的接口	n	HTTP	该接口功能为服务路由功能节点注册、认证信息交互、系统中各功能节点网络拓扑状态及使用状态信息交互
内容服务控制与节点管理功能模块之间的接口	o	HTTP	接口功能为内容服务控制功能节点注册、认证、资源和使用状态信息交互

3.3　CDN 部署架构

节点是指多台物理设备在某地理区域范围内作为一个整体对外提供内容服务。每个 CDN 节点通常包含多台服务器设备，节点是 CDN 系统最基本的组成单元。

CDN 系统设计的主要目标是尽量提高用户请求的响应速度，为达到这一目标，CDN 节点部署的原则是尽量将内容存放在最靠近用户的位置。通俗地说，就是将为用户提供实际内容服务的服务器部署在网络的边缘位置。中心节点层保存着完整的内容副本，当用户请求的内容在边缘层未命中时，中心层可能会为用户直接提供服务，也可能由下级边缘节点向中心节点请求拉取内容，再分发到边缘节点为用户提供直接服务。但是，当用户请求量过大时，

若大量边缘节点都直接向中心节点请求拉取内容，会造成中心层压力过大，这时，就需要考虑在边缘节点层和中心节点层之间部署一个区域层。区域层保存了部分内容副本，使其能够分发内容并在边缘节点未命中时提供服务，以减轻中心节点的压力。这样，就形成了 CDN 的三级系统架构。图 3-3 是一个典型的三级架构 CDN 系统部署示意。

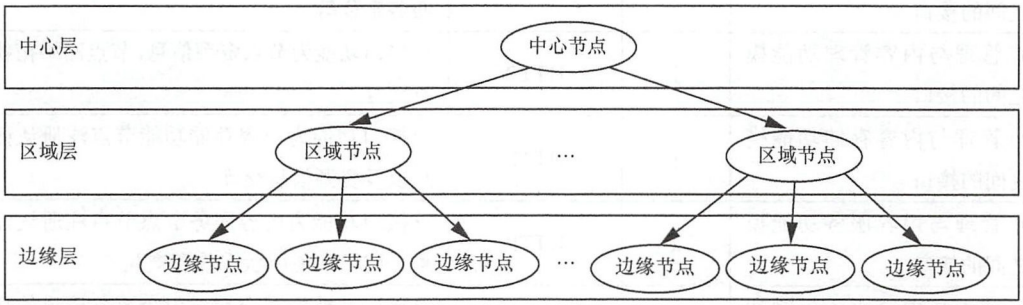
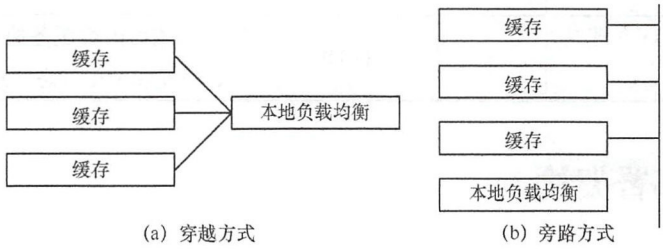


图 3-3　三级架构 CDN 系统部署

从节点构成的角度来说，无论是 CDN 区域层节点还是 CDN 边缘层节点，都是由缓存设备和本地负载均衡设备（SLB）构成的。在一个 CDN 节点中，缓存设备和本地负载均衡设备的连接方式有两种：一种是旁路方式，另一种是穿越方式，如图 3-4 所示。



(a) 穿越方式　　(b) 旁路方式

图 3-4　CDN 节点内缓存和 SLB 的连接方式

在穿越方式下，SLB 一般由四七层交换机实现，SLB 向外提供可访问（公网）的虚拟 IP（VIP）地址，每台缓存设备分配不同的私网 IP 地址，该 SLB 连接下挂的所有缓存设备构成一个服务单元。所有用户请求经由该 SLB 设备，再由该 SLB 向上向下进行转发。SLB 实际上承担了网络地址转换（Network Address Translation，NAT）功能，向用户隐藏了各台缓存设备设备的 IP 地址。这种方式是 CDN 系统中应用较多的方式，优点是具有较高的安全性和可靠性，但是，当节点容量大时，四七层交换机容易形成性能瓶颈[3]。

在旁路方式下，SLB 有两种实现方式。在早期，SLB 一般由软件实现。SLB 和缓存设备都具有公共的 IP 地址，SLB 和缓存是并联关系。用户需要先访问 SLB 设备，然后再以重定向的方式访问特定的缓存。这种实现方式简单灵活，扩展性好，缺点是安全性较差，而且需要依赖于应用层重定向。随着技术的发展，四七层交换机也可采用旁路部署方式，旁挂在路由交换设备上，数据流量通过三角传输方式进行转发[3]。

3.4　CDN 间组网

当 CDN 覆盖范围或能力不足，或需要多厂商时，CDN 可以进行组网。不同 CDN 的共同组网目标是实现 CDN 分发与服务能力的共享，各 CDN 通过标准接口实现互联互通。

CDN 共同组网根据服务的场景及各 CDN 的功能与性能不同，可选择不同的组网架构，典型的组网逻辑可分为以下两种。

（1）并联组网

源站同时接入多个 CDN，通过用户请求调度层面进行流量分配，不同 CDN 共同承载内容，如图 3-5 所示。

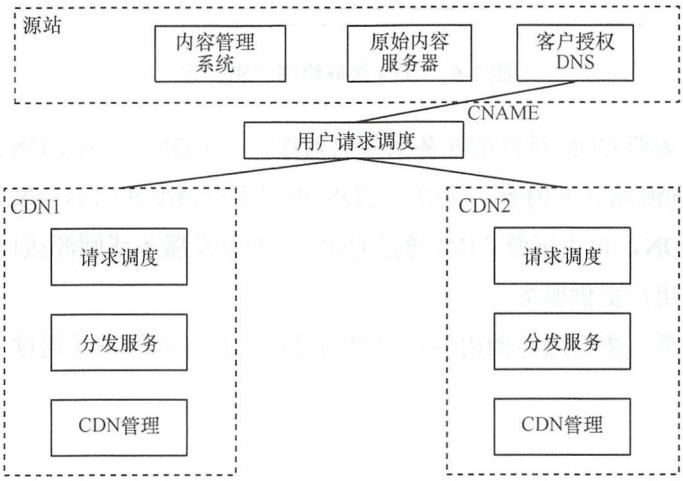


图 3-5　CDN 并联组网逻辑示意

并联组网方式需要把用户流量通过 CNAME 引导到一个用户请求流量调度系统，由该调度系统把请求分配至不同 CDN。不同 CDN 间不进行内容的分发与服务互联，均需与源站系

统进行互联的实现内容注入，或分别回源获取内容，再独立进行分发服务。在一个区域内引入多家 CDN 服务提供商向用户提供 CDN 服务时，一般采用这种组网方式。

（2）级联组网

源站接入上游 CDN，上游 CDN 再进一步和下游其他 CDN 对接，上游 CDN 和下游 CDN 除调度层面外，CDN 内容分发与服务层面也进行互联，共同组成一张统一的 CDN，如图 3-6 所示。

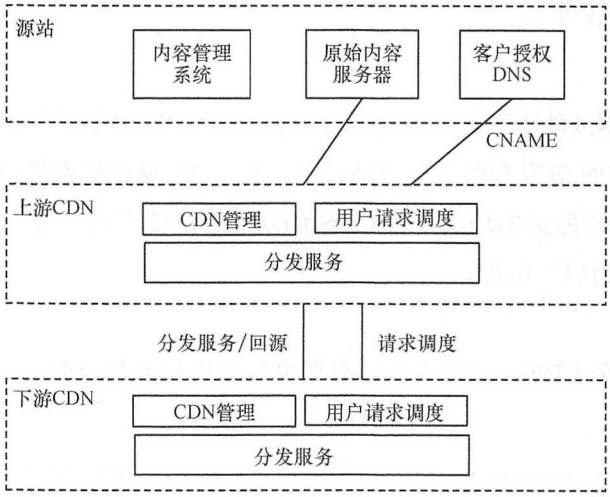


图 3-6　CDN 级联组网逻辑示意

级联组网方式需要 CDN 承载的业务系统只对接一个 CDN（上游 CDN），向该 CDN 注入内容或由该 CDN 向源站获取内容，并由该 CDN 决定用户调度和内容分发策略，把用户请求调度到其他下游 CDN，再由下游 CDN 通过 CDN 间的分发服务或回源接口实现上下游 CDN 间的互联，向最终用户提供服务。

为保证服务质量，需要服务的内容也可以通过内容预注入的方式通过上游 CDN 提前注入下游 CDN 中。

第4章

CDN关键技术

由CDN的工作过程和CDN的架构可以看出，使用CDN的业务流程与未使用CDN的流程相比，增加了一些步骤，感觉业务流程变复杂了，其实，终端用户使用CDN服务，是完全不需要用户参与的。从用户的角度来看，完全感觉不到是否使用了CDN进行访问加速，但却做到了为用户请求找到一个最优的服务器提供服务，从而保证了用户访问的质量。通过CDN的工作过程和CDN的架构，可以归纳出下面这些CDN使用的关键技术[1]。

（1）内容统一ID

CDN不是为单个CP/SP或单个业务服务的。CDN RR需要对接多个门户系统，这些门户系统的业务ID是没有统一规定的，也没有办法统一规定。如何解决内容统一编码是CDN在总体架构设计时首先要考虑的问题。

内容统一编码的解决办法总体而言有以下两种思路。

- 内容在进入CDN前进行统一的编码，即所有由CDN承接加速的业务都应按预先制定好的规律进行统一的编码。使用这种方式编码的局限性很明显：对于托管到CDN中的不同的业务内容，很难由一个外部业务系统来统一进行编码。
- CDN不要求内容预先统一ID的编码，而是对每个接入的业务或CP/SP内容增加唯一标识。这个标识是CDN统一内部编码的，可以保证在CDN的唯一性，当然每个业务或CP/SP的内容ID也是唯一的。

（2）负载均衡技术

在 CDN 系统中，负载反映服务器当前的健康状态，包括 CPU、内存和磁盘使用情况以及网络使用情况以及缓冲区的大小、用户连接的数量和其他信息。负载均衡是指根据一定的策略将服务器的请求分配给多台服务器，以获得最佳的服务性能。从广义上讲，负载均衡可以分为本地负载平衡（SLB）和全局负载均衡（GSLB）。本地负载均衡是指 CDN 节点内多台服务器间的负载均衡，全局负载均衡是指分布在各个 CDN 节点之间的负载的分摊与平衡，通常被用来作为用户请求路由调度中处理节点满负荷的一种策略；本文后面提到的负载均衡主要指本地负载均衡。

（3）用户请求路由调度

用户请求路由调度是指根据一定的调度策略，把用户请求路由或重定向到合适的 CDN 节点的过程。用户请求路由调度技术一方面要能根据用户的特征准确地判断用户所属的地理位置，尽量把用户调度到最靠近用户的 CDN 节点进行服务，另一方面，需要实时监控服务器集群的健康状况，以避免将请求调度到 CDN 节点中不可访问的边缘节点[4]。根据网站提供的不同服务类型，主要有基于 DNS 的重定向或基于 HTTP 的重定向策略。

（4）内容缓存技术

CDN 内容缓存技术是指根据一定的策略和规则，在 CDN 节点的本地服务器上存储热点内容。缓存技术使得用户不需要再向源站请求相关内容，从而减少了源站的访问压力。CDN 的这种把内容存储在最靠近用户的缓存服务器的方式，在一定程度上说明 CDN 是基于缓存技术开发并通过分布式服务器集群实现的。

（5）内容分发技术

内容分发作为 CDN 的核心机制，负责从 CDN 的中心节点下发文件到每个区域和边缘节点。CDN 的高效分配指的是利用有限资源最大化地实现用户就近服务。

4.1 统一内容 ID

CDN 需要依靠内容 ID 来定位内容，并为用户提供服务。CDN 的内容 ID 的唯一性很重要，因为如果出现了重复的内容 ID，CDN 将会把错误的内容发送给用户。

4.1.1　统一资源定位符

当看到统一内容 ID 时，相信很多读者会想到统一资源定位符（Uniform / Universal Resource Locator，URL）。URL 是 Internet 上标准的资源地址（Address）。它最初是由 Tim Berners Lee 发明用来作为万维网的地址的，现在已经被万维网联盟编制为 Internet 标准 RFC1738。URL 使用广泛，而且有一套机制保证其不会重复[1]。

互联网上的每一个网页、每一个资源都具有唯一的名称标识，通常称为 URL 地址。URL 一般由几部分组成：协议类型、主机名和路径及文件名。

- 协议：指定使用的传输协议。协议类型主要有：HTTP、FTP、Gopher、Telnet、File 等[1]。
- 主机名：是指存放资源的服务器的域名系统 （DNS）主机名或 IP 地址。有时，在主机名前也可以包含连接到服务器所需的用户名和密码（格式：username:password）。
- 端口号：整数，是可选参数，省略时使用方案的默认端口，各种传输协议都有默认的端口号，如 HTTP 的默认端口为 80。如果输入时省略，则使用默认端口号。有时候出于安全或其他考虑，可以在服务器上对端口进行重定义，即采用非标准端口号，此时，URL 中就不能省略端口号这一项。
- 路径：由零或多个“/”符号隔开的字符串，一般用来表示主机上的一个目录或文件地址。
- 内容 ID：由字符串组成，一般用来表示该路径下的唯一文件标识。

除了以上部分的内容外，URL 还可以包括参数：这是用于指定特殊参数的可选项。如查询参数“?query（查询）”，用于给动态网页（如使用 CGI、ISAPI、PHP/JSP/ASP/ASP.NET 等技术制作的网页）传递参数，可有多个参数，用“&”符号隔开，每个参数的名和值用“=”隔开。

4.1.2　CDN 内容统一 ID

前文介绍了统一资源定位符（URL），读者自然会想到，直接使用 URL 作为 CDN 内容的统一 ID 不就是最好的方案吗？

许多 CDN 确实是直接使用 URL 作为其 CDN 的内容 ID 的。这带来的好处是很明显的：CDN 不再需要进行设置，由 URL 的唯一性特性来保证 CDN 内容 ID 的唯一性。但这种方法有以下两个主要缺点[1]。

- 内容 ID 没有可以被 CDN 利用来进行分类的信息，即内容 ID 没有 CDN 可利用的信息。在实际的应用中，CDN 对内容加速时，如果能根据内容的一些属性提供一些特殊的功能，对 CDN 总体性能十分有用。
- 不适合没有使用 URL 标识的内容。在许多应用场景，内容接入 CDN 是通过内容管理系统注入的，往往这些内容只在内容管理系统内标识是唯一，没有使用 URL 作为标识。

所以 CDN 内容统一 ID 单纯使用 URL 标识并不是一个很好的办法。考虑到 CDN 需要支持多源 CMS 和在同一 CMS 上存在多屏内容注入的需求，可考虑引入二元组 UniContentID 来唯一标识不同 CMS 注入的不同内容域的唯一的内容文件。以下为一种 CDN 内容统一 ID 方案。

UniContentID 可定义为二元组（ProviderID，ContentID），是 CDN 全网中进行全局调度的唯一内容标识。ProviderID 定义为每个 CMS 或业务系统的不同域内容提供商在整个 CDN 中的唯一标识，需要在接入点进行配置，并在 CDN 全网配置相关的 URL 匹配规则和 ProviderID 属性[1]。对于同一个 ProviderID 的不同内容，用不同命名的 ContentID 进行区别。值得注意的是，不同 ProviderID 会出现相同命名 ContentID。因此，在 CDN 全网 UniContentID 二元组（ProviderID，ContentID）才能表示为唯一内容标识。

4.2 本地负载均衡

4.2.1 负载均衡技术

负载是指 CPU 使用、内存使用、网络负载、可用缓冲区、应用系统负载、用户数以及其他各种系统资源的当前状态信息。负载均衡是指将服务器的负载信息，进行分析决策后，进行动态分配网络流量，从而使各个服务器的负载趋于相对平衡的一种策略。通俗地说，负载均衡功能类似于轮流值班，将任务分配给每个人，这样就不会使人辛劳过度。在这里，负载均衡设备充当了一种转发功能，类似于路由器。

同一 CDN 节点中，通常也会包含多个服务器设备。当大量用户访问时，可能会出现节点内的单个服务器被分配大量繁重的内容分发任务的情况，此时，在处理能力和吞吐量等方面会出现严重的性能瓶颈。为了实现服务器的最佳性能，负载均衡技术经常会被使用。

（1）负载均衡的指标选取

首先，要选取衡量服务器负载情况的指标，通常有内存利用率、网卡流量、硬盘 I/O 吞吐量、服务成功率、在线用户数以及响应时间等。

理想的负载指标应满足低测量开销的要求，并且各负载指标在测量及控制上彼此独立。然后，利用一定手段探测获取服务器的负载指标值，并分析是否达到负载均衡的状态。

（2）负载状况的获取

需要通过探测来获取负载状况，主要有以下 3 种技术手段。

- 使用代理技术——标准的网络管理协议（SNMP）：SNMP 是管理进程和代理进程之间的通信协议。在所需服务器上使用 SNMP 功能，可以探知服务器的准确负载状况。SNMP 的优点在于所有主流厂商都支持，并且所有 SNMP 管理的设备使用相同的管理接口以支持通用的管理消息集合，但却不一定能满足定制化负载信息参数的探测。
- 使用第三方的软件：有一些第三方开发的软件可以完成负载探测工作。例如，Nagios 作为一款开源监控软件，能够对系统的 CPU、磁盘、网络负载等参数进行探测，且配置灵活。
- 使用自己开发的软件：最简单的方法是自己开发负载均衡软件来检测服务器的负载状况。

总结来说，负载均衡的关键是资源使用，其最终目标是使每个服务器的资源利用状态尽量达到平均。所以，准确把握各个服务器的负载情况，并动态调整流量分配是负载均衡技术的核心。

4.2.2　负载均衡的技术分类

为满足不同场景的应用需求，目前存在许多不同的负载均衡技术。如基于 DNS 的负载均衡、基于客户端的负载均衡和基于 OSI 模型的负载均衡等。

（1）基于 DNS 的负载均衡

作为最先出现的负载均衡技术，该方法通过为多个域名配置同一个 IP 地址，使得不同用户请求同一个域名时，可以访问不同的服务器来实现。

（2）基于客户端的负载均衡

这种方法是指在客户端执行特定程序，实时采集服务器的负载状况，然后，按照一定策

略，找到可以提供服务的最佳服务器并发起请求。

这种方法要求每个客户端程序都拥有服务器集群的某些知识，然后利用负载均衡方式向不同的服务器发送请求。例如，在客户端中运行美国伯克利大学研发的 JavaApplet 程序，JavaApple 向每个服务器都发起负载信息采集请求，然后将这些信息返回给用户。但其缺点是透明性不理想，并且每个客户端程序都需要具有集群功能，不具有普遍的适用性。

（3）基于 OSI 模型的负载均衡

OSI 模型在其第二、第三、第四和第七层均具有相应的负载均衡策略，具体如下。

• 基于数据链路层（二层）的负载均衡

数据链路层位于 OSI 模型的第二层，二层负载均衡通常使用基于 MAC 地址的虚拟 MAC 地址模式来分摊负载。用户对外部虚拟 MAC 地址发起请求，负载均衡设备接收后，为后端分配实际的 MAC 地址响应。二层负载均衡技术会提供一个外部 VIP（虚拟 IP）地址，并且，后台服务器的实际 IP 地址是相同的，实际内部 MAC 地址各不相同。那么，服务器 IP 地址相同会产生 IP 地址冲突吗？答案是否定的。因为这些服务器都比较特殊，不发送 ARP 报文，并且要求负载均衡设备连接交换机的端口也不发送 ARP 报文。只要负载均衡设备发出报文，交换机根据 MAC 地址表项转发即可。

• 基于网络层 （三层）的负载均衡

网络层处在 OSI 模型的第三层中。一般地，三层负载均衡采用虚拟 IP 地址，是基于 IP 地址来实现分摊流量的。负载平衡设备提供了一个外部 VIP（虚拟 IP）地址，但后台服务器之间的实际 IP 地址是各不相同的。客户端请求虚拟 IP 地址，负载平衡设备接收后，分配实际 IP 地址响应（每个服务器对应不同的实际 IP 地址）。

• 基于传输层（四层）的负载均衡

传输层位于 OSI 模型的第四层，也称为四层负载均衡技术。四层负载均衡是以三层负载均衡为基础的，通过三层的虚拟 IP（VIP）地址，再添加四层的端口号来实现的，即用 IP 地址+端口接收请求，再转发到对应的服务器。在三层负载均衡的基础上加上端口号的目的地址，用来决定哪些流量需要进行负载均衡处理。处理这些流量并转发到后端服务器，然后记录 TCP 或 UDP 流量是由哪个服务器处理的，后续将这个连接的所有的流量也转发到同一服务器进行处理。LVS 就是一个常见的传输层开源负载均衡工具。

• 基于应用层（第七层）的负载均衡

应用层在 OSI 模型的第七层，也称第七层负载均衡，就是基于虚拟 URL 等应用层信息

的负载均衡，例如，客户端通过请求虚拟 URL，分配给用户真正提供服务的服务器。七层负载均衡的优点是使整个网络变得更加智能。例如，当通过七级负载均衡方式访问网站时，对于图片类请求能够被转发到特定缓存服务器，并可用缓存技术实现负载均衡；对于文本类请求可通过特定的文本压缩技术，减轻负载。当然，这只是七层负载均衡应用的两个小例子，从技术原理上讲，七层负载均衡能够极大地提高网络层应用系统的灵活性。常见的应用层开源负载均衡工具有 Nginx 和 Apache。

总之，二层负载均衡为：收到一个对虚拟 MAC 地址的请求，然后分配给用户实际 MAC 地址的服务器以提供服务；三层负载均衡为：收到一个对虚拟 IP 地址的请求，然后分配给用户实际 IP 地址的服务器以提供服务；四层负载均衡为：一个虚拟 IP 地址+端口的方式，确定什么样的流量需要进行负载均衡，并为用户分配实际的服务器以提供服务；七层负载均衡为：接收对虚拟 URL 的请求，然后分配给用户实际提供服务的设备以提供服务。所谓 4~7 层负载均衡，就是在需要对后台服务器进行负载均衡的背景下，根据四层信息或七层信息来确定转发流量的策略。

4.2.3　负载均衡的算法

负载均衡算法是负载均衡技术的核心内容。负载均衡的研究分为两个方向，即静态负载均衡和动态负载均衡。静态负载均衡是采用某种分配算法在任务执行前即确定分配到各个服务器的方案，其分配基于系统平均情况，不考虑系统瞬时状态变化，基于对负载的计算量、通信关系和依赖关系以及服务器集群本身的状况等先验知识或预测形成远程执行进程表。动态负载均衡可根据当前运行状态自适应决定负载均衡策略，动态方法是通过集群系统的实时负载信息，动态地将负载在各个服务器之间进行分配和调整。

考虑到服务请求的不同类型、服务器的不同处理能力、网络状况以及随机选择造成的负载分配不均匀等问题，为了更加合理地把负载分配给内部的多个服务器，就需要应用相应的、能够正确反映各个服务器处理能力及网络状态的负载均衡算法，下面介绍一些常用的负载均衡算法[1]。

（1）轮询（Round-Robin）算法

假设所有服务器处理性能相同，将外部请求按顺序轮流分配到集群中的服务器上。这种算法的优点是具有简洁性，无需记录当前所有连接的状态，是一种无状态算法，但是不适用

于服务器组中处理性能不一的情况，而且当请求服务时间变化较大时，容易导致服务器间的负载不平衡[1]。

（2）加权轮询（Weighted Round-Robin）算法

为保证处理能力强的服务器处理更多的访问流量，用相应的权值表示服务器的处理性能，将请求数目按权值的比例分配给各服务器。负载均衡设备可以自动询问服务器的负载情况，并动态地调整其权值。这种均衡算法也是一种无状态算法，但可以解决服务器间性能不一的情况，能确保高性能的服务器得到更多的使用率，避免低性能的服务器负载过重[1]。

（3）随机（Random）均衡算法

把来自网络的请求随机分配给各个服务器。

（4）加权随机（Weighted Random）均衡算法

此种均衡算法类似于加权轮询算法，不过在处理请求分担时是个随机选择的过程。

（5）最小连接（Least-Connection）算法

该算法是一种动态算法，通过服务器中当前所活跃的连接数来估计服务器的负载情况，把新的连接请求分配到当前连接数最小的服务器。但是，当各个服务器的处理能力不同时，该算法并不理想。

（6）加权最小连接（Weighted Least-Connection）算法

用相应的权值表示各个服务器的处理性能，具有较高权值的服务器将承受较大比例的负载。负载均衡设备可以自动问询服务器的负载情况，并动态地调整其权值。

（7）目标地址散列（Destination Hashing）算法

根据请求的目标 IP 地址，将其作为散列键（Hash Key），通过散列（Hash）函数将这个目标 IP 地址映射到一台可用且未超载的服务器，将请求发送到该服务器，属于静态映射算法。

（8）源地址散列（Source Hashing）算法

与目标地址散列算法相反，根据请求的源 IP 地址，作为散列键，通过散列函数将请求的源 IP 地址映射到一台可用且未超载的服务器，将请求发送到该服务器。除了将请求的目标 IP 地址换成请求的源 IP 地址，该算法采用的散列函数与目标地址散列算法相同，算法流程与目标地址散列算法基本相似。在实际应用中，源地址散列和目标地址散列算法可以结合使用在防火墙集群中，它们可以保证整个系统有唯一出入口[1]。

（9）基于局部性的最少连接（Locality-Based Least Connection）算法

找出请求的目标 IP 地址最近使用的服务器，若该服务器是可用的且没有超载，将请求发

送到该服务器；否则用“最少连接”的原则选出一个可用的服务器。算法的设计目标是在服务器的负载基本平衡情况下将相同目标 IP 地址的请求分配给同一台服务器，提高各台服务器的访问局部性和主存（主要缓存设备）命中率[1]。

（10）带复制的基于局部性最少连接（Locality-Based Least Connections with Replication）算法

与基于局部性的最少连接算法的不同之处在于，它要维护从一个目标 IP 地址到一组服务器的映射，而不是从一个目标 IP 地址到一台服务器的映射。该算法找出请求的目标 IP 地址对应的服务器组，按“最小连接”原则从服务器组中选出一台服务器，若服务器可用且没有超载，将请求发送到该服务器；否则按“最小连接”原则从这个集群中选出一台服务器，将该服务器加入服务器组中，再将请求发送到该服务器。另外，若该服务器组有一段时间未被修改，则将最忙的服务器从服务器组中删除，以降低复制的程度[1]。

还有一些其他的负载均衡算法，在此不一一列举。总的来说，各种算法都有它的优点和缺点。负载均衡效果越好，算法越复杂，考虑的因素越多，附加的开销就越大。在能满足负载分配均衡要求的情况下，应尽可能使用简单的算法，以提高负载均衡的工作效率，避免瓶颈效应。

4.3　用户请求路由调度/全局负载均衡

CDN 中的用户请求路由（Request Routing，RR）调度，也就是通常所说的全局负载均衡（Global Server Load Balancing，GSLB），是指 CDN RR 调度设备根据预先设定的策略把用户请求路由或调度到引导到最佳的边缘节点访问，从而保证某一地区的用户请求能够就近访问和快速响应。用户请求路由的设计涉及就近性判断、路由调度机制和流量分配策略等问题。就近性判断就是 CDN 调度服务器需要知道用户所在的区域，从而可以把用户调度到最近的 CDN 节点。路由调度机制就是 CDN RR 调度服务器通过什么样的技术手段把用户访问请求引导到指定的 CDN 节点。

用户请求路由调度一方面要能根据用户的特征准确地判断用户所属的地理位置，另一方面还要实时监测各地服务器集群的健康性，避免请求被重定向到无法访问的边缘节点[4]。根据网站提供的服务类型，目前比较常用的方法有基于 DNS 的重定向和基于 HTTP 的重定向。

4.3.1 基于 DNS 的用户调度

（1）智能 DNS 技术

域名系统（Domain Name System，DNS）是负责完成域名与 IP 地址相互转换的网络基础设施。

DNS 的结构就像一棵倒挂着的树，如图 4-1 所示。DNS 的根节点以“.”表示。每个节点也是一个新的子树状结构的根节点，而这些子树状结构称为 DNS 中的一个域。每个域具有其独特的名称，由域名便可识别其位置，类似于文件系统的绝对路径。在 DNS 中，域名是从该域的根节点到整个树状结构的根节点，其间一连串以“.”隔开卷标的组合。

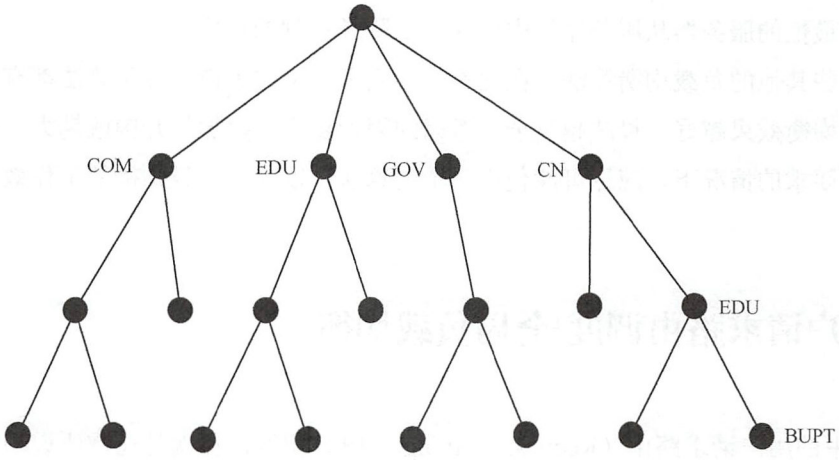


图 4-1 DNS 域名结构关系

DNS 系统，是由本地 DNS（Local DNS，LDNS）、授权 DNS 以及配套的网络接入设备组成的。本地 DNS 负责为互联网用户通过域名访问网络时提供解析服务，它通过缓存已解析域名的结果，提供更快的用户请求响应。用户在终端上配置的 DNS 服务器地址其实就是本地 DNS 的服务地址。授权 DNS 负责互联网业务的授权域名数据，提供授权域名的地址解析服务。

当用户请求访问一个域名时，若本地 DNS 缓存中存在该域名的解析记录，则本地 DNS 立刻向用户返回该域名解析记录；否则，本地 DNS 需要通过递归查询，依次访问各级授权 DNS 服务器，以获得此域名权威的 DNS 解析记录，再将其发送给用户，同时，本地 DNS 会

将得到的解析记录缓存下来，当用户下一次发起请求时，直接将该解析记录返回给用户端。具体域名解析递归如图 4-2 所示。

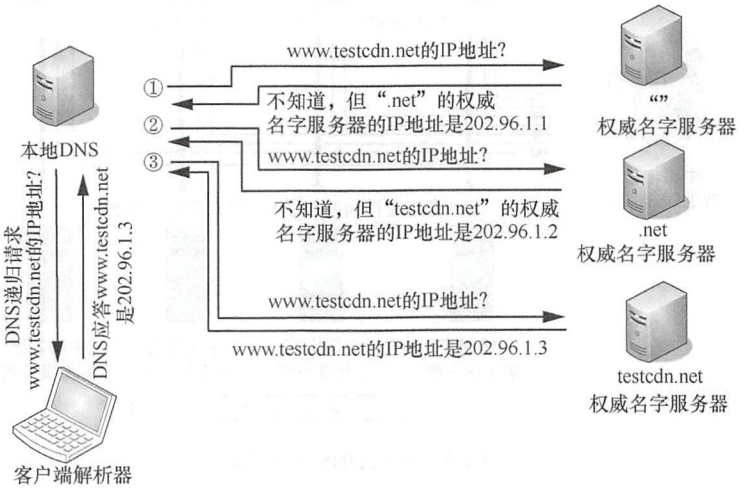


图 4-2　DNS 域名解析递归示意

普通的 DNS 服务器只负责为用户解析出 IP 地址记录，而不去判断用户从哪里来，这样会造成所有用户都只能解析到固定的 IP 地址上。

智能 DNS 会判断用户的来历，智能地判断访问网站的用户，做出一些智能化的处理，根据不同的访问者把域名分别解析成不同的 IP 地址，然后把智能判断后的 IP 地址返回给用户。

智能 DNS 策略解析技术能根据用户的地区来源，分城市或地区来判断用户来源，并且能够自动判断用户的上网线路是上海电信还是广东电信，然后智能返回对应的上海电信和/或广东电信服务器的 IP 地址。如图 4-3 所示，智能 DNS 会自动判断用户的上网线路所属的运营商，然后返回用户所在上网线路的运营商服务器的 IP 地址。

（2）基于 DNS 重定向调度

当用户向域名发起访问请求时，经过 DNS 解析后，向用户返回了该域名对应的 IP 地址，用户通过与该 IP 地址建立 TCP 连接，最终获得网站的内容。

引入 CDN 后，首先需要在域名的 DNS 服务器上增加一条 DNS 记录，即域名的 DNS 记录指向位于 CDN 的 GSLB 设备，这样对所有域名的解析将由 CDN 的 GSLB 设备负责。下面具体介绍基于 DNS 重定向的调度流程，如图 4-4 所示。

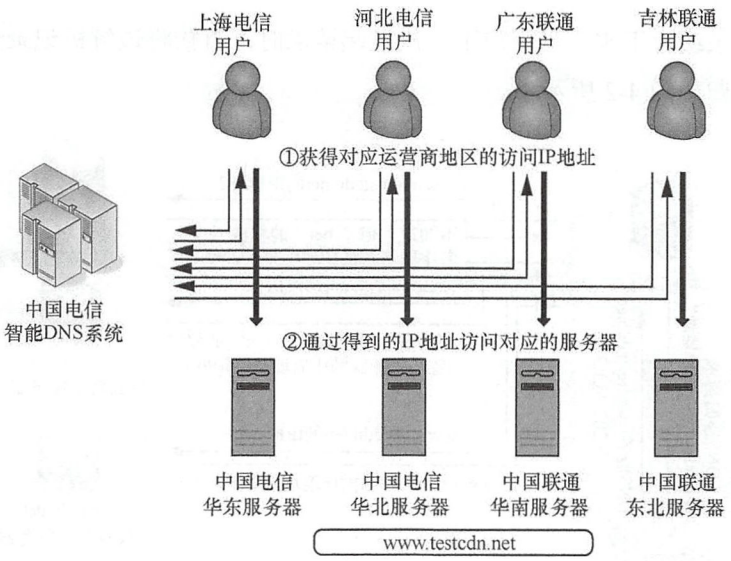


图 4-3　智能 DNS 解释流程

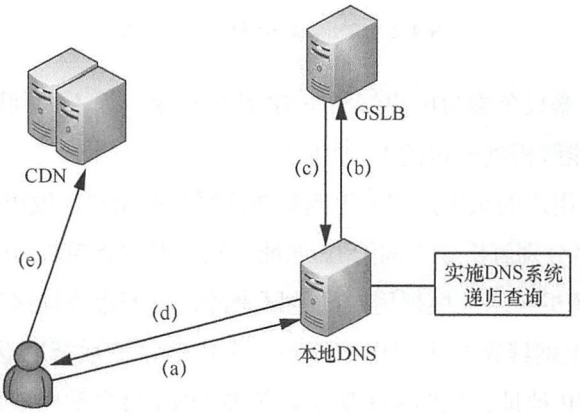


图 4-4　基于 DNS 重定向的用户调度流程

（a）用户向本地 DNS 发起域名解析请求。

（b）本地 DNS 服务器通过一系列的 DNS 查询得知授权解析服务器是 GSLB 设备，向 GSLB 发起域名解析请求。

（c）GSLB 设备综合分析负载、内容分布以及链路状况等信息，找出一个离本地 DNS 最近的健康 CDN 节点，并返回给本地 DNS。

（d）本地 DNS 传给用户获得域名解析后提供服务的该 CDN 节点的 IP 地址。

（e）用户对返回的 CDN 节点的 IP 地址发起 TCP 连接请求，该 CDN 节点最终为用户提供内容服务。

如果在 DNS 重定向中，加入智能 DNS 技术，就能自动判断用户的上网线路是中国联通还是中国电信，向用户返回广东电信 CDN 节点的 IP 地址（比如，用户是处于广东地区的中国电信用户），以此来实现将用户请求调度到最佳的 CDN 节点。

（3）策略的优劣分析

DNS 重定向技术易于实现，用户调度只发生在域名解析的阶段，该方式的特点在于整个就近性判断和重定向过程都发生在用户请求域名解析时，而非用户真正请求服务器连接时，而且与请求的应用层协议无关，并支持主流的网络应用层协议[1]。该策略的优点在于：

- 实现简单，用户请求路由简捷，一般用来做省市级别和跨运营商的“粗粒度”调度；
- 并不依赖于分发内容类型及相应的协议，无论是基于 Web 还是 MMS、RTSP 等分发都毫无影响[1]。

在 CDN 的全局负载均衡方案的应用中，如果只采用 DNS 流量分配方式，则存在着如下问题[1]：

- DNS 解析记录信息可能被整个 DNS 解析过程中的任何一个 DNS 服务器缓存，会导致全局的 DNS 流量分配失败，因为这个 DNS 服务器不会再将 DNS 的解析请求转发给 WSD-NP，从而用户将不能得到最新的最优站点的 IP 地址；
- 就近性判断是对用户本地 DNS 服务器的 IP 地址进行的，而不是用户本身的 IP 地址；如果用户客户端设定的本地 DNS 服务器不是距离用户最近的 DNS 服务器，则用户无法得到就近服务；
- 无法向支持直接给出 IP 地址的业务提供 CDN 服务。

4.3.2　基于 HTTP 的重定向

（1）HTTP 重定向基本原理

应用层重定向主要利用了 HTTP、MMS、RTSP 等协议本身的重定向机制来实现，由于各种应用层协议的重定向机制基本相同，因此，以 HTTP 为例来介绍重定向功能。

在 HTTP 中，存在 301（永久重定向）、302（暂时重定向）与 meta fresh（特定时间后重定向）3 种重定向方式，其作用都是将 CDN 用户请求调度转移到另一台服务器设备上，通常

采用的是 302 重定向。

比如，浏览器请求 www.testcdn.net 这个域名，服务器返回的 302 响应消息中包含了重定向的 URL 地址。浏览器得到这个响应后，会向重定向得到的新 URL 再次发起请求，由此，真正获得了网页内容。

（2）基于 HTTP 重定向调度

下面介绍基于应用层重定向的 GSLB 的完整工作流程，如图 4-5 所示。

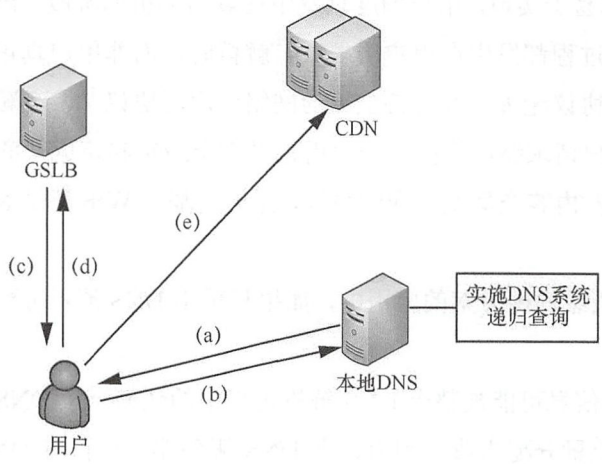


图 4-5　基于 HTTP 重定向示意

（a）用户向本地 DNS 发起域名解析请求；

（b）经过本地 DNS 系统实施递归查询后，向用户返回 GLSB 设备的 IP 地址；

（c）用户向该 GSLB 发起 HTTP 请求，请求该网站的内容；

（d）GSLB 设备综合分析用户 IP 地址、负载、内容分布以及链路状况等信息，通过 HTTP 302 重定向后选择一个最适合的 CDN 节点返回给用户；

（e）用户向该 CDN 节点发起 HTTP 访问请求，该 CDN 节点最终为用户提供内容服务。

（3）策略的优劣分析

该方式的特点在于可以根据用户的真实 IP 地址进行就近性判断，该策略的优点如下：

- 与智能 DNS 方式相似，RR 在做 HTTP 重定向时，也会根据链路质量、用户所在位置、服务器健康状态选择合适的服务节点。区别在于，相比较于 DNS 重定向方式，基于 HTTP 重定向方式的 RR 系统能够看到用户发起请求的 IP 地址以及用户的具体请求，

利用用户真实的 IP 地址归属地信息，比 DNS 重定向的判断更加准确，可以真正做到根据“网络的就近性”分配流量，能够进行更准确的定位和“精细粒度”的调度策略和流量控制，这是 HTTP 重定向方式的最大优点；

- 与基于 DNS 重定向的策略可以无缝结合，实现合理、完善内容请求路由；
- 支持不同的 ICP 接入 CDN 的业务模式，服务割接非常快捷。

该方式的不足在于[1]：

- 由于 HTTP 重定向过程需要额外解析域名，还需要与 URL’建立 TCP 连接并且发送 HTTP 请求，使得响应时间加长；
- 不同于 DNS 方式，没有任何用户请求能被外部系统终结，所有请求都必须进入 GLSB 系统，这将成为性能和可靠性的瓶颈；
- 由于此方式的就近性判断与重定向是当用户与核心节点 VIP 建立连接时发生的，因此对于相应的分发内容类型和相关协议有所限制，例如，针对 HTTP 可以采用通用的 302 协议重定向，但对于类似于 MMS 这类私有协议就无法通过通用做法来实现；
- 随着访问用户的增加和网络规模的扩张，核心调度点的压力会较大，因此核心点设备的冗余设计变得非常关键。

4.4　内容缓存技术

4.4.1　缓存技术

CDN 缓存是指在一定时间内按一定规则保存在某一物理设备或节点上的内容，缓存的内容可能是文档、照片、视频及其他文件等。

CDN 实现缓存功能的服务器通常就叫缓存服务器，通常 CDN 缓存功能模块与 CDN 流媒体服务器合设。当用户首次向缓存服务器发起访问请求时，由于缓存服务器的内容未命中，则需要向上级服务器发起内容请求拉取内容，为用户提供“边拉边放”的服务。缓存服务器根据一定的缓存算法，决定对下载下来的文件进行保存或删除。由于缓存服务器离用户很近，所以响应速度快，能够节省传输成本，并且减少了源站服务器的压力。

按照不同的需求，缓存服务器的工作模式可以分为正向代理和反向代理两类。在正向代

理工作模式下，用户需要在客户端配置正向代理的地址，此举会给用户造成一定麻烦；而反向代理则无需进行地址配置，并且用户也感觉不到代理的存在，客户端会按照 DNS 解析的结果，直接访问代理。反向代理作为 CDN 最常采用的方式，可以被用于加密、缓存、压缩以及防攻击。

4.4.2 缓存替换算法

由于 CDN 服务器存储空间有限，所以不可能把所有资源都放在一个缓存服务器上，需要对缓存的内容进行定时更新，把用户访问最多的内容保存下来，而删除其余的内容。缓存算法就是对这些服务器所缓存的内容进行选择和更新。缓存算法的意义在于，根据用户的请求习惯，更新缓存中的数据可以提高用户请求的命中率，或者从另一种角度看，缩短了用户请求的整体响应时延，极大地提高了请求高峰时间的网络能够承受的流量。一个好的缓存算法，不仅重视命中率，还考虑相应的存储成本、运行效率等众多因素。总之，缓存算法是基于用户请求情况，对缓存中的数据进行筛选，其目的在于提高响应用户请求的效率。

早期的缓存技术可用来节省带宽以减少网络拥塞，但它们不可避免地引起了以下问题：一是用户有可能得不到网页的及时更新，因为缓存区不可能自动跟踪网页的变化；二是为得到最新网页，用户访问时需要首先查询真正的服务器上的内容，这将导致访问速度的降低。因此，需要对缓存算法进行改进。

缓存算法发展至今，已有一些效果令人较为满意的成果，从一开始包括基于访问频率的算法到基于访问时间的算法再慢慢完善，现在已有了访问时间与访问频率相结合等较为理想的算法。基于访问频率的算法，通过在某段时间内对资源被访问的次数进行统计，以此来判断该资源接下来会不会被访问。这类算法理所当然，最容易理解，因此也最容易被接受，包括 LFU（Least Frequently Used，最少频率使用）以及在此基础上发展起来的 2Q（2 Queues）、LIRS（Low Inter-Reference Recency Set）等；基于访问时间的算法，通过记录资源访问的时间，以时间作为判断依据，包括 LRU（Least Recently Used，最近最少使用）；访问时间与访问频率相结合的算法，综合了基于访问频率和访问时间的算法，包括 LRFU（Least Recently Frequently Used）、FBR（Frequently Based Replacement）等[1]。当然，因为现实中有各种各样的因素，这些缓存算法各自或多或少地存在一些优点和缺点，因此，没有一种缓存算法能完美解决所有的缓存问题。结合实际情况，通过对已知的缓存算法进行改善，使之能切合我们

的目标，才是关键所在。

目前 CDN 缓存算法主要分为访问频率、访问时间以及由此衍生出的变种算法，下面将介绍这几种算法[1]：

- “最近最少使用”算法；
- “最近最少使用”算法的变种；
- “最少频率使用”替换算法；
- 基于分数因子的缓存算法；
- 基于块等级缓存算法；
- 基于流行度的块等级缓存算法。

（1）LRU 算法

LRU 算法，顾名思义，缓存中将保留最近一段时间内经常使用的数据，而淘汰最近未被经常使用的数据。LRU 基于这样一个事实：在最近一段时间内经常被使用的数据在未来一段时间内也会被使用，而未被经常使用的数据在未来很长时间内不会被用到。因此，在替换内容时只需要找出最近最少使用的那些数据进行替换即可[1]。

LRU 在缓存中维持一个内容列表，存储最近被使用的数据。当有数据被请求时，它将跃至第一位。如果被请求的数据在列表中，即已经在缓存中存储（比如排在第 k 位），那么排在 k 之前的数据位置将拉到向后一位，没有内容会被淘汰。如果被请求的数据不在列表中，它将被提取到缓存中，跃至第一位的同时，其他内容的位置顺次拉后一位，排在最后一位的内容将被淘汰。

例如，假设有 1、2、3、4、5 共 5 个内容，底层服务器只够容纳 3 个内容，初始时为 1、2、3 这 3 个内容。第一个用户请求内容 2，内容 2 在服务器中存在，内容次序改为 2、1、3；第二个用户请求内容 4，内容 4 在缓存中不存在，缓存向高级服务器拉取内容 4，并将内容 4 提至第一位，次序变为 4、2、1，原来的内容 3 将被淘汰。

“最近最少使用”算法遵循这样一个原则：最近被访问的内容在未来被访问的几率增大，于是才出现了经典的“最近最少使用”算法。“最近最少使用”算法的提出是显然的，它基于内容访问热度来对内容进行取舍，留下访问次数多的，剔除访问次数少的，这符合多数人的设想。然而，不难看出，该算法具体的操作过程仍然需要研究和改进。经典的“最近最少使用”算法单纯地从最近的访问次数出发，直接将最近一次被访问的内容的优先级提至最高。然而，最近被访问的内容不一定是最热门的，这将导致非热门的内容次序较高，影响用户访

问内容和服务器响应用户的效率。因此，从这种角度上来看，这种算法的效率很低，需要对其进行改进。

（2）LRU 算法的变种

LRU 算法的一个最大缺陷是将最近访问的内容的次序直接提至第一位，这是不合理的，因为最近被使用的内容却不一定是最受欢迎的，提至第一位的做法略为不妥。于是，从这个角度出发，可以对该算法进行一些改进，因此产生出了 LRU 算法的变种。

在 LRU 算法中，每当用户访问一个内容时，该内容的次序提至第一位，而在该算法的变种中，被访问的内容的次序将不再被提至第一位而是被提至第 J 位，排在第 J 位之后的内容次序依次增加，末位遭到替换，如图 4-6 所示。

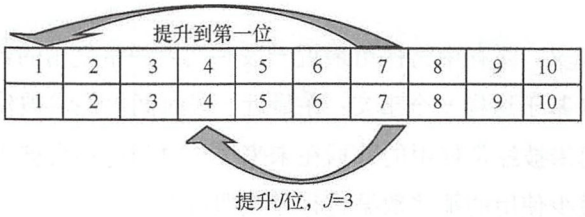


图 4-6　LRU 算法示意

如图 4-6 所示，在 LRU 算法中，原来第 7 位的内容由于最近依次被访问导致它被提至第一位，但是该内容有可能在未来一段时间内不被访问，因此，对其作出这样的改动：该内容提升 J 位（J=3），从而降低因为它不被访问而占用首位资源造成的低效率。这样做可在某种程度上提升 LRU 算法的效率，毕竟这使得 LRU 算法不再那么紧张，在内容排序上有了一定舒缓的空间，最近依次被访问的内容有了更多的上升空间，其他未被访问的内容也有更多变化的可能[1]。

（3）LFU 算法

在“最近最少使用”算法及其变种中，最近访问的时间是考虑的重要因素。而“最少频率使用”算法从频率上出发，按照内容访问频率对内容进行排序，这比单纯地考虑访问时间要合理得多，毕竟，访问频率才是能够反映内容热度的一个重要标准。在以后的很多算法中也采取了跟访问频率相关的方法。另外，“最少频率使用”算法也使用了不同于“最近最少使用”算法的缓存机制[1]。

在“最少频率使用”算法中，缓存一般分为两部分：一部分是固定缓存 AC（Actual Cache，

实际缓存），一部分是隐藏缓存 SC（Shadow Cache，影子缓存）。固定缓存，是更新不频繁、资源变动较少的那部分缓存；隐藏缓存，则是更新频繁、资源变动较大的那部分缓存。当用户发出访问请求时，未命中的内容将先被存储到 SC 中。在一定时间间隔 T 内，缓存中会维持一个频率列表，记录每个内容被访问的次数。T 时间过后，{AC,SC}中被请求次数最多的内容将被更新到 AC 中，而下一轮 T 之内，AC 内的内容不会被替换掉。

理论上来看，“最少频率使用”算法从内容的访问频率上对内容进行排序，留下访问频率高的，替换掉访问频率低的，而因为访问频率跟内容热度有很大的关联，因此，热度高的内容将会有很大的几率被保存下来，这正是缓存算法优化的目的。

“最少频率使用”算法从另一个角度出发，考虑了内容被访问的次数，即内容被访问的频率，显然这是符合常理的，能够反映内容的热门程度。另外，AC 和 SC 的增加使得“最少频率使用”的准确率得到提高，这是该算法的优势之一。然而，“最少频率使用”算法也有其难点所在，比如计算频率的时间段 T 的控制、AC 和 SC 容量的控制、替换的时间设定等。总之，在不同场合，该算法都需要得到不同的改进。

（4）基于分数因子的缓存（Scoring based Caching）算法

最简单的基于分数因子的缓存算法，是对每一个内容（v_i），都维护一个分数（s_i）。每一个内容 i 在被放进缓存时，都会得到一个初始化的分数，比如 $s_i=B$。每一次内容 i 被请求，该内容都得到一个新的分数，而其余内容的分数也会相应调整，比如 $s\_i=s_i=s_i+A$，而其余内容的分数 $s_k=s_k-1$（$k\neq i$）。这是一种简单的基于分数因子的缓存算法，在维护一个分数的基础上，有以下两种替换策略[1]。

- 第一种，区间分数法，定义一个有效分数区间[M, N]，若某部不在缓存中的内容的分数，达到了这个区间，即 $M<s_i<N$，则把这部视频放到缓存中；若某部已在缓存中的内容的分数，离开了这个区间，即 $s_i<M$ 或 $s_i>N$，则在缓存中删除该内容。
- 第二种，最优分数法，根据一个节点的最大的存储能力，如 L 个内容，选择最优分数的内容进行缓存。每次有一个内容的分数有改变，则进行排序，凡是分数不在前 L 的内容从缓存中删除，凡是分数在前 L 的内容加入缓存中。

在以上规则下面，每一个对 CDN 内容的请求，会出现以下 3 种处理事件。

- 请求的内容已经在缓存中，即请求“命中（hit）”。在这种情况下，对内容的分数作出调整，但无需在缓存中剔除任何内容，也没有必要向上级节点请求内容。在这种情况下，CDN 内部没有流量消耗。

- 请求的内容不在缓存中而算法要求节点缓存该内容，因为算法使得该内容在被请求后达到了某个进入的分数，如区间$[M,N]$或前L名，则需要在缓存中剔除掉原来的第L个内容（若需要），并且向上级节点复制被请求的内容，然后在复制内容之后向请求用户发送被请求的内容。在这种情况下，CDN内部有流量消耗，即边缘节点向中心节点复制了内容。
- 请求的内容不在缓存中而算法不要求节点缓存该内容，因为该内容没有进入区间或前L名。在这种情况下，依然调整该内容的分数，但是不需要从中心节点复制任何内容，即CDN内部没有流量消耗。

一般来说，节点应该维护每一个内容的分数s_i（每一部是指出现在CDN系统内且允许用户请求的内容）。但是实际上，分数过低的内容，一般就不维护它的分数，将它从关注列表中移出。

由于记录内容分数及其他存储需要，每个节点的存储能力都将分为两个部分，较大的一部分用来存储内容，较小的一部分用来存储列表、分数等与管理相关的内容。存储管理相关内容的存储空间，会与内容的空间严格分开，因此从管理者的观点看去，节点有一部分存储能力“消失”了。这部分“消失”的存储能力，成为影子缓存。

事实上，绝大多数复杂的缓存算法，都是基于分数因子的。分数因子提供了一个对内容重要性（或者说受欢迎程度、未来需求量）的一个评价，依据这个评价可以选择是否缓存或者剔除一个内容。因此，基于分数因子的缓存算法不是一个实践中独立的算法，而是其他算法的一个基础或者框架。

（5）基于块等级的缓存（Chunk-Level Caching）算法

基于块等级的缓存算法，是在上述分数因子的缓存算法基础上，把每个内容分成大小相同、内容连续的K块分别存储，然后每个内容块的分数s_k继承自原内容的分数s_i。定义第i个内容的第k内容块的分数为$s_{i,k}$。

每次内容i被访问，则对内容i内的所有块，均有$s_{i,k}=s_{i,k}+1$。每次视频i的第k块被用户完整访问，则对该块的分数有$s_{i,k}=s_{i,k}-1$。如果一个用户停止访问某内容块（例如，对视频内容进行停止观看包括“快进”和“快退”等操作，表明用户对该块不感兴趣），则分数$s_{i,k}$要依据以下规则进行改变：所有k之后的块需要执行以下操作$s_{i,j}=s_{i,j}-1$（$k<j<K$）。

大部分时候，块等级的缓存算法，与基于分数因子的缓存算法十分相像。不同的地方在于之前是通过比较s_i来进行取舍，现在是比较$s_{i,k}$。

基于块等级的缓存算法，主要用于单个内容较大的内容分发网络，如视频分发网络。它考虑了这样一个事实，即人们可能对内容的某一部分感兴趣，但把整个内容存储下来会产生浪费[1]。例如，对于视频内容，人们只浏览了视频的开头，就会选择看或者不看。因此，在计分规则中，充分考虑了这些事实：

- 一个用户请求了某个内容，则可能对这个内容的其他部分都感兴趣；
- 一个用户访问过某一块，则可能不会再次访问了；
- 一个用户停止访问某内容块，则可能不会再访问这个内容块之后的其他内容块。

（6）基于流行度的块等级缓存（Popularity based Chunk-Level Caching）算法

这是一个仅考虑流行度的块等级缓存算法，即计分方式是统计每个块被访问的次数，属于一个较简单的变种。

总的来说，缓存算法的意义在于尽快满足用户的访问，即希望用户的请求尽可能多地利用节点现有内容提供，而不需要向上级节点复制内容。因此，缓存算法的优劣应该看对“未来”访问的命中率。可以定义一个未来时间段 t，假设有 M 次访问到达，而这些访问命中缓存的次数是 N，则 t 时间内的命中率为 $p=N/M$。若 t 和 p 都较大，表明在较长的时间内，命中率较高，则这是一个好的缓存算法。

4.5　流媒体技术

4.5.1　实时流媒体技术：RTSP

近年来，视频业务的快速发展，使得视频内容流量已经占到整个互联网流量的一半。说到互联网视频，不得不提到流媒体技术，是它促进了当前互联网视频的快速发展。传统媒体内容分发技术主要有两大类[5]。一类是以基于 RTCP（Real-time Transport Control Protocol，实时控制协议）、RTP（Real-time Transport Protocol，实时传输协议/实时传输协议）的实时流媒体技术为代表的，还有一类是目前主流的视频网站使用的 HTTP 渐进式下载。本节首先介绍实时流媒体技术[5]。IP 网络的实时流媒体数据传输协议通常包括 RTP、RTCP 和实时流协议（Real-time Streaming Protocol，RTSP）。这些协议协同合作完成了流媒体控制、流媒体传输、流媒体质量报告等工作，实现了 IP 网上的可靠流媒体传输。这几种协议之间的关系可参考图 4-7。

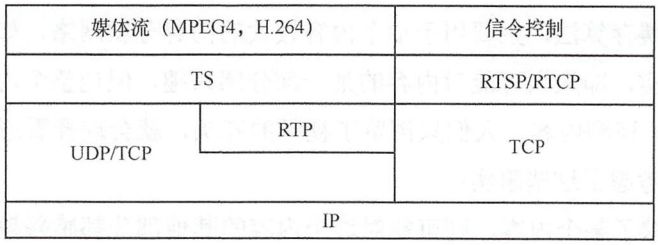


图 4-7　协议之间的关系

如图 4-7 所示，RTSP 作为一个流媒体信令控制协议，能够有效地通过 IP 网络传送多媒体数据。RTSP 在体系结构上位于 RTP，并使用 TCP 或 RTP 实现数据传输功能。RTP 是媒体流传输协议，负责把流媒体加上协议报文，便于后续终端进行处理。RTCP 则提供了流量控制和拥塞控制服务。

（1）RTP 和 RTCP

RTP 是在 Internet 上针对多媒体数据流的一种传输协议，工作于一对一或一对多的传输情况，可提供时间信息和实现流同步。RTP 通常使用 UDP 来传送数据，也可在 TCP 之上工作。当应用程序开始一个 RTP 会话时，会使用到两个端口，一个给 RTP，一个给 RTCP。RTP 本身并不能为按顺序传送数据分组提供可靠的传送机制，也不提供流量控制或拥塞控制，而是依靠 RTCP 提供这些服务。通常 RTP 算法并不作为一个独立的网络层来实现，而是作为应用程序代码的一部分[1]。

RTCP 与 RTP 共同提供流量控制和拥塞控制服务。在 RTP 会话期间，参与者周期性地传送 RTCP 分组，这些分组中含有已发送数据分组的数量、丢失数据分组的数量等统计数据，服务器可根据这些信息动态地改变传输速率，甚至改变有效载荷类型[1]。RTP 与 RTCP 的配合使用可有效地进行反馈，从而减小开销，提高传输效率，非常适合传送网上的实时数据。

（2）RTSP

RTSP 是由哥伦比亚大学、Netscape 公司和 RealNetworks 公司联合提出的一种基于 TCP/IP 的应用层协议，相关 IETF RFC 标准参见 RFC2326。它定义了如何使一对多应用程序有效地通过 IP 网络传送多媒体数据。RTSP 在体系结构上位于 RTP、RTCP 之上，它使用 TCP 或 RTP 完成数据传输。RTSP 用于具有实时性的流媒体播放控制，它在客户端和流媒体服务器之间建立起实时会话，客户端可以发起播放、暂停、快进、快退、退出等控制命

令与服务器交互，而在客户端发起连接请求到连接中断的整个过程中，服务器会一直监听客户端的状态，持续不断地发送媒体数据分组，保证传输质量。RTSP 对流媒体提供了诸如暂停、快进和快退等控制，而它本身并不传输数据，RTSP 的作用相当于流媒体服务器的远程控制[1]。传输数据可以通过传输层的 TCP、UDP，RTSP 也提供了基于 RTP 传输机制的一些有效的方法。

RTSP 一般与 RTP/RTCP 和 RSVP 等底层协议一起协同工作，提供基于 Internet 的整套的流服务。它可以选择发送通道（例如：UDP、多播 UDP 和 TCP）和基于 RTP 的发送机制。它可以应用于多播和点播。RTSP 标准规范中定义的方法见表 4-1。

表 4-1　RTSP 标准规范中定义的方法

方法	方向	要求	说明
ANNOUNCE	客户端→服务器	可选	客户端发起声明，将媒体对象描述发送给服务器
	服务器→客户端	可选	服务器实时更新连接描述
DESCRIBE	客户端→服务器	推荐	获得并检查媒体对象的描述信息，包括 URL、应答类型等信息
OPTIONS	客户端→服务器	推荐	可任意时刻发出 OPTIONS 请求，获取可用方法
	服务器→客户端	可选	
PAUSE	客户端→服务器	推荐	暂停媒体播放，临时中断流传输，但仍然保持客户端和服务器连接，收到 PLAY 请求后继续传输
PLAY	客户端→服务器	推荐	客户端通知服务器按照 SETUP 规定的方式开始传输数据，默认情况下 PLAY 请求从媒体流的起始位置开始传输，若 PLAY 请求携带了 range 参数可以指定起始位置
SETUP	客户端→服务器	推荐	建立流媒体传输机制，包含 URL 地址、端口及其他参数
REDIRECT	服务器→客户端	推荐	服务器将客户端重定向到另一服务器地址
TEARDOWN	客户端→服务器	推荐	请求中止流媒体传输，解除客户端和服务器的连接状态

抓取分组软件能够展示 RTSP 请求实例，客户端和服务器之间通过 OPTIONS、DESCRIBE、SETUP 等方法进行交互。终端与 CDN 节点间的 RTSP 播放信令可参见图 4-8。

RTSP 目前主要应用于对实时性要求较高的流媒体服务，如 IPTV 点播、直播、时移、回看业务，由于客户端和服务器需要一直保持连接状态进行信令交互和流传输，对服务器性能及网络质量也有较高要求。

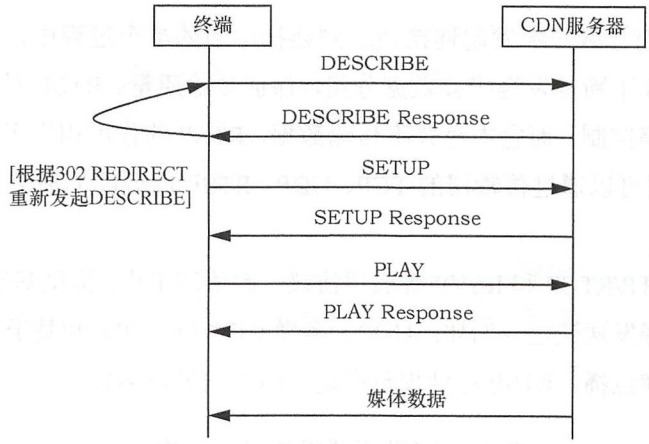


图 4-8　终端与 CDN 节点间的 RTSP 播放信令

4.5.2　渐进式下载流媒体技术：HTTP Streaming

第 4.5.1 节中提到，传统的内容服务技术主要有两大类，一类是以 RTSP/RTP 为代表的实时流媒体技术，而另一类则是目前主流视频网站采用的 HTTP 渐进式下载，本节从 HTTP 入手，介绍 HTTP 渐进式下载流媒体技术[1]。

（1）HTTP

HTTP 是一个属于应用层的面向对象的协议。它于 1990 年被提出，经过几年的使用与发展，得到不断的完善和扩展。

HTTP 的主要特点可概括如下[1]。

- 支持客户/服务器（C/S）模式。
- 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。
- 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。
- 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
- 无状态：HTTP 是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状

态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

关于 HTTP 还涉及以下几个重要概念[1]。

- 连接（Connection）：一个传输层的实际环流，它建立在两个相互通信的应用程序之间。
- 消息（Message）：HTTP 通信的基本单位，包括一个结构化的八元组序列并通过连接传输。
- 请求（Request）：一个从客户端到服务器的请求信息包括应用于资源的方法、资源的标识符和协议的版本号。
- 响应（Response）：一个从服务器返回的信息包括 HTTP 的版本号、请求的状态（例如“成功”或“没找到”）和文档的 MIME 类型。
- 资源（Resource）：由 URI 标识的网络数据对象或服务。
- 实体（Entity）：数据资源或来自服务资源的回应的一种特殊表示方法，它可能被包围在一个请求或响应信息中。一个实体包括实体头信息和实体的本身内容。
- 客户机（Client）：一个为发送请求目的而建立连接的应用程序。
- 用户代理（User Agent）：初始化一个请求的客户机。它们是浏览器、编辑器或其他用户工具。
- 服务器（Server）：一个接受连接并对请求返回信息的应用程序。
- 源服务器（Origin Server）：是一个给定资源可以在其上驻留或被创建的服务器。
- 代理（Proxy）：一个中间程序，它可以充当一个服务器，也可以充当一个客户机，为其他客户机建立请求。一个代理在发送请求信息之前，必须解释并且最好可以重写它。代理经常作为通过防火墙的客户机端的门户，代理还可以作为一个帮助应用来通过协议处理没有被用户代理完成的请求。
- 网关（Gateway）：一个作为其他服务器中间媒介的服务器。与代理不同的是，网关接受请求就好像对被请求的资源来说，网关就是源服务器；发出请求的客户机并没有意识到它在同网关打交道。网关经常作为通过防火墙的服务器端的门户，网关还可以作为一个协议翻译器以便存取那些存储在非 HTTP 系统中的资源。
- 通道（Tunnel）：两个连接中继的中介程序。一旦激活，通道便被认为不属于 HTTP 通信，尽管通道可能是被一个 HTTP 请求初始化的。当被中继的连接两端关闭时，通道便消失。当一个门户（Portal）必须存在或中介（Intermediary）不能解释中继的通信时

通道被经常使用。

- 缓存（Cache）：反映信息的局域存储。

（2）RTSP 与 HTTP 的比较

RTSP 负责在服务器和客户端之间建立并控制一个或多个时间上同步的连续流媒体，其目标是类似 HTTP 为用户提供文字和图形服务那样，为用户提供连续媒体服务[1]。因此，RTSP 的设计在语法和操作上与 HTTP 很相似，这样，对于 HTTP 的大部分扩展也适用于 RTSP。但是 RTSP 和 HTTP 在很多方面有着区别[1]。

- HTTP 是一个无状态协议，而 RTSP 是有状态的。
- HTTP 本质上是一个非对称协议，客户端提出请求而服务器响应；而 RTSP 是对称的，服务器和客户端都可发送和响应请求。
- 在 RTSP 中，每个演示（Presentation）及其所对应的媒体流都有一个 RTSP URL 标识。整个演示及媒体特性都在一个演示描述（Presentation Description）文件中定义，该文件可能包括媒体编码方式、语言、RTSP URL、目标地址、端口及其他参数。用户在向服务器请求某个连续媒体流的服务之前，必须首先从服务器获得该媒体流的演示描述文件以得到必需的参数，演示描述文件的获取可采用 HTTP、E-mail 或其他方法。
- RTSP 中的所有操作都是通过服务器和客户端的消息应答来完成的，其消息包括请求（Request）和响应（Response）两种，RTSP 正是通过服务器和客户端的消息应答来完成媒体流的创建、初始化（SETUP）、VCR 控制（PLAY、PAUSE）以及拆线（TEARDOWN）等操作的。在基于 C/S 结构的分布式视频点播系统中，客户机在向视频服务器请求视频服务之前，首先通过 HTTP 从网页服务器获取所请求视频服务的演示描述文件，利用该文件提供的信息定位视频服务地址（包括视频服务器地址和端口号）及视频服务的编码方式等信息。然后客户机根据上述信息向视频服务器请求视频服务。视频服务初始化完毕，视频服务器为该客户建立一个新的视频服务流，客户端与服务器运行 RTSP，以对该流进行各种 VCR 控制信号的交换，如播放（PLAY）、停止（PAUSE）、快进、快退等。当服务完毕，客户端提出拆线（TEARDOWN）请求，需要说明的是，服务器使用 RTP/UDP 将媒体数据传输给客户端，一旦数据抵达客户端，客户端应用程序即可播放输出。在流式传输中，使用 RTP/RTCP/UDP 和 RTSP/TCP 两种不同的通信协议在客户端和服务器间建立联系。

（3）HTTP 渐进式下载

HTTP 渐进式下载（Progressive Download），本质上就是标准的 HTTP（RFC2616）。具体原理是客户端向服务器发起视频点播请求后，服务器通过 HTTP 方式向客户端传输视频内容，传输速度取决于客户端侧、网络侧和服务器侧的网络带宽，当客户端下载的视频内容达到一定大小时（例如下载到了视频内容中的关键帧）即开始播放，播放时仍然继续下载后续的内容，从而实现边下载边播放的功能。以前国内外众多主流的视频网站均采用了这一方式提供服务，其视频封装格式通常为 flv、mp4、ts、f4v 等。HTTP 渐进式下载有以下一些特点[1]。

- 部署非常简单，通过开源的 Apache、Lighttpd、Nginx 等 HTTP 服务器即可轻松部署，而客户端只要支持 HTTP 标准协议即可。
- HTTP 是无状态的协议，服务器只是被动地接受客户端请求，将请求数据传输给客户端，并不主动检测客户端的状态，每次 HTTP 连接都是一个独立的会话，HTTP 可看作“单向”，而 RTSP 可看作“双向”。

HTTP 标准规范中定义的方法见表 4-2[1]。

表 4-2　HTTP 标准规范中定义的方法

名称	功能
GET	发起服务请求，向服务器获取 URI 中标识的相关资源，服务器在返回响应请求时会包含该资源
POST	向服务器提交数据处理请求，该数据已被包含在消息体中
HEAD	功能与 GET 类似，但服务器在返回响应请求时不包含资源，只返回相关报头，可应用于只需获取报头信息而不需要传输整个响应请求的内容时
PUT	向服务器提交数据上传请求
DELETE	向服务器提交数据删除请求
TRACE	向服务器请求返回响应请求的信息，用于故障诊断和定位
CONNECT	指向一个能将连接改为管道方式的代理服务器
OPTIONS	向服务器请求返回针对特定资源所支持的 HTTP 请求方法

抓取分组软件能够展示典型的 HTTP 渐进式下载请求实例，客户端通过 GET、POST 等方法向服务器发起请求，而服务器只被动地返回 200 OK 响应。

4.5.3 自适应流媒体技术

前面介绍了媒体内容分发技术，即 RTSP 和 HTTP Progressive Download，本节介绍基于 HTTP 的另一种流媒体协议，即 HAS（HTTP Adaptive Streaming，自适应流媒体）。自适应流媒体技术，是指通过智能感知用户的下载速度，动态调节媒体内容的编码速率，提供高质量、平滑的媒体内容播放的体验。通过融合传统 RTSP/RTP 流媒体技术和基于 HTTP 渐进式下载技术，使得自适应流媒体技术拥有高效性、可扩展性以及兼容性强的特点。作为混合的媒体分发方式，HAS 技术以流的形式为用户带来体验，并且采用与渐进式下载相同的 HTTP 进行内容下载分发，然而，不同之处在于，这些媒体内容都被分割成了一系列的切片进行内容传输[5]。

HTTP 自适应流媒体技术的基本原理是在媒体内容在编码阶段，将同一个片源切割成码率不同的分片（如标清分片、高清分片）。对于同一个片源切割成的一个个分片内容，用户需要对每一个内容分片分别发起一次 HTTP 请求，用户的客户端能够根据网络带宽状况实时调整 HTTP 请求，在带宽不稳定的情况下，获取不同码率的内容分片，实现基于带宽变化的码率自适应。

分片切割是 HAS 技术的关键，通常每个分片时长为 2～10 s，且时间长度相同。这表明了每个分片都是由许多完整的视频 GOP 构成的，每个分片都需要包含关键帧（I 帧），以此来确保每个分片都是独立的。

媒体分片一般存储在网站的服务器中，客户端向网站服务器请求媒体分片，并用 HTTP 方式下载媒体分片，当分片下载到客户端时，客户端会根据正确的播放顺序，播放这些内容分片。这些分片之间，内容不重叠且无缝隙衔接，因此，用户能够获得平滑流畅的播放体验。

互联网上，现在主流的 HTTP 自适应流媒体技术主要有以下 3 种：苹果公司的 HTTP Live Streaming（HLS）流媒体技术、微软公司的基于 Silverlight 的 HTTP Smooth Streaming（HSS）平滑流媒体技术和 Adobe 公司基于 Flash 的 HTTP Dynamic Streaming（HDS）动态流媒体技术。这 3 家公司为 HTTP 自适应流媒体技术提供了相应的解决方案，包括编码、切片、CDN 分发以及用户服务等一系列环节。对于上述自适应流媒体技术，如图 4-9 所示。

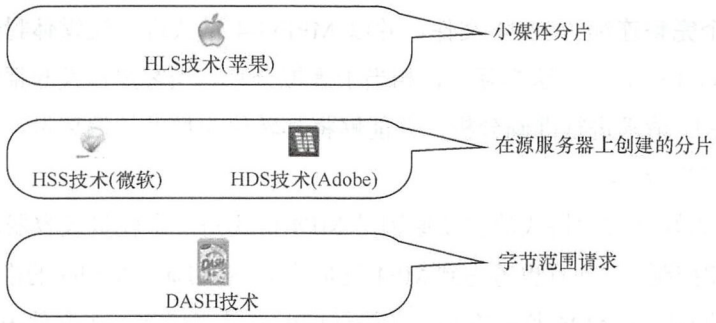


图 4-9　主流的自适应流媒体技术

此外，OIPF、MPEG 以及 IETF 等国际标准组织也提出了相应的 HTTP 自适应流媒体技术标准，例如，DASH 标准与苹果、微软以及 Adobe 公司的基础思路一致，也是通过感知用户下载速度来自适应动态调整视频的播放。目前，MPEG DASH 标准工作组，正在融合各厂商已经实现的自适应流媒体特点，进一步推进自适应流媒体技术的标准化发展。

（1）HLS

苹果公司的 HLS 自适应流媒体技术设计的最初目的是通过普通的网站服务器将视频内容推送到苹果终端，例如，iPhone、iPad 以及苹果的笔记本电脑和台式机。

服务器部件、分发部件和客户端构成了 HLS 的技术方案。由编码器接收媒体内容的输入，并利用 H.264 技术编码，输出 MPEG-2 TS 媒体流[5]。输出的媒体内容文件是基于 MPEG2 的一系列 TS 流分片，每个 TS 分片单独存放在服务器上。在切片过程中，按照预先设定的时间间隔切割媒体内容并保存为一个个小 TS 分片。同时还创建了这些 TS 分片的 m3u8 索引文件，该索引文件包含了分片顺序的列表和每个分片的具体信息[5]。

在媒体内容传输过程中，客户端会向服务器请求并下载 m3u8 索引文件，通过解析 m3u8 文件得到媒体内容的分片信息。m3u8 索引文件具有二级层次架构，在索引文件起始处使用 #EXTM3U 标签以示区别。在一级列表中，利用#EXT-X-STREAM-INF 标签提供各码率的媒体内容列表地址，该地址能够链接到二级 m3u8 索引文件上。而二级 m3u8 列表包含了所有同一码率的媒体内容切片的时长和地址。

（2）HSS

微软公司提供的 HTTP 自适应流媒体解决方案是 HTTP 平滑流媒体技术，是以微软公司的头端服务 IIS 7 和其终端的 Silverlight 技术为基础实现的。在平滑流媒体技术中，将一个完

整片源存储为一个完整连续的 MP4 文件，并以 MPEG-4 格式进行流媒体封装，将每个分片封装成一个 MPEG-4 的分片。这实际上，相当于虚拟分片。当客户端发起播放请求时，头端服务器需要对 URL 请求进行准确分析，并能够将其转化为精准的偏离量，以此能够找到对应的分片分发给客户端[5]。

选择 MP4 作为媒体文件格式的主要原因是 MP4 格式是一个轻量级容器，能够使用.NET 方便地进行管理和控制[5]，并且也考虑到 MP4 还是基于 ISO Base Media 的格式规范，被广泛应用。而最关键的原因是 MP4 格式在最开始设计的时候就考虑在一个文件内实现媒体内容分片。对于 HTTP 平滑流媒体技术的存储媒体格式和传输媒体格式分别如图 4-10 和图 4-11 所示。

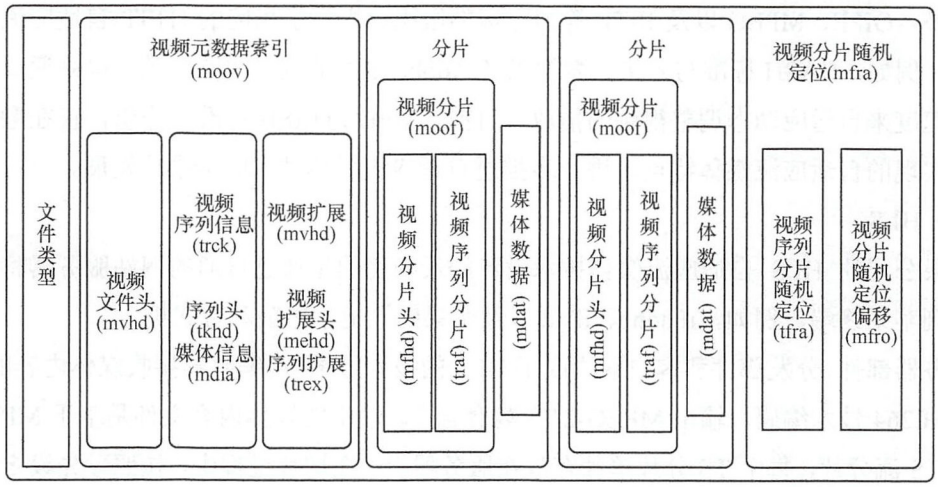


图 4-10　微软平滑流媒体技术存储媒体格式

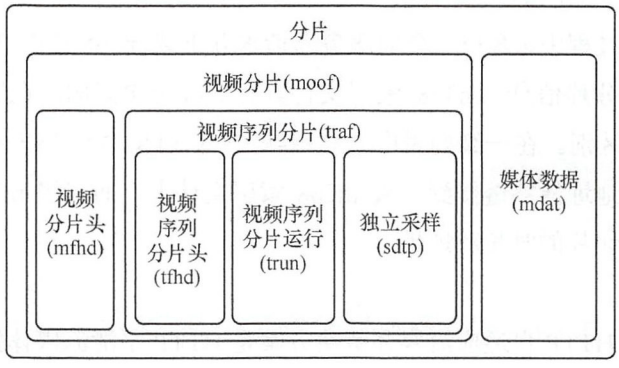


图 4-11　微软平滑流媒体技术传输媒体格式

可以看出，平滑流媒体技术采用的是虚拟切片方法，并没有对媒体内容进行真正切片，其存储仍然是一个完整文件的形式，但在播放过程中，每个分片会按照终端的请求独立分发给客户端[5]。

平滑流媒体技术是终端基于 Silverlight 实现的，包括了对 MPEG-4 格式的解析、HTTP 请求下载以及码率的切换[5]。并且，微软公司以.NET 代码的形式将上述功能开放给开发者进行调用，进行播放器的效果优化和调整，而码率切换模块是开发工作中最核心和最复杂的部分。

（3）HDS

HDS 技术是 Adobe 公司对苹果公司的自适应流媒体技术和微软公司的平滑流媒体技术的一个回应。通过 HTTP 网络连接分发媒体内容与适应用户当前带宽的内容传输能力是这 3 种自适应流媒体技术的相同之处。但是，与接入以太网的个人电脑相比，有效带宽对于 Android 终端便显得十分不足，因此，HDS 技术实现的最终目的是，无论是用户使用手机终端、个人电脑还是其他终端设备，都能实现媒体文件的流畅播放体验。

Adobe 公司的传统流媒体技术，是采用 RTMP+FLV 的组合来实现的流媒体方案，得到了互联网视频行业的广泛应用[5]。为了满足码率动态自适应的需求，Adobe 公司在其传统流媒体解决方案上进行码率自适应优化，后来，便推出了基于 HTTP 的码率自适应动态流媒体技术。

HDS 技术是由多个部件协同工作来实现的，并通过 HTTP 将媒体内容发送给客户端。HDS 架构如图 4-12 所示，内容制作模块包括点播和直播的处理模块。点播内容打包模块将一个完整的媒体文件进行分片，并用 F4F 的格式存储[5]。直播打包模块将直播流实时写入 F4F 媒体文件当中[5]。源站（HTTP 源模块）负责存储 F4F 媒体文件和对应的 F4M 格式的索引文件，并把编码、分辨率以及码率等信息存放在 F4M 索引文件中。

HDS 的优势在于将 RTMP 和 HTTP 的优点结合到了一起。因为使用传统的 HTTP“渐进下载”进行在线播放时，必须等到下载完成才能观看，但是，下载完成的媒体文件能够缓存在客户端。而使用 RTMP 能够让媒体内容分割成多个数据分组并源源不断从服务器传输到客户端，因此，客户端可以在媒体内容的任意一个时间点请求传输内容，请求到哪里就从哪里开始下载内容，但是，观看完毕之后媒体文件无法缓存在客户端中。

结合上述两种协议的特点，HDS 技术通过对来自 RTMP 端的“流”进行包装处理，然后，转化成 HTTP“流”在客户端解析，使得用户既能够在媒体内容的任一时间点请求内容播放，不用等到完整的媒体文件下载完毕，又能够使得媒体文件缓存在客户端，而不用下一次播放再重新向服务器发起请求。

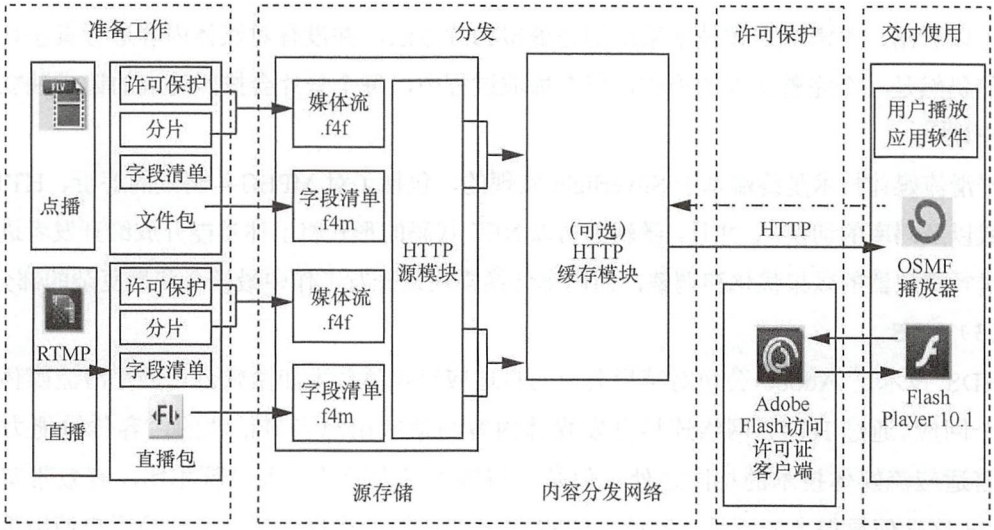


图 4-12　HDS 架构

HDS 有按需分配和实时处理两种工作模式。在按需分配模式下，直接对文件进行“流”处理，把单个文件分离成许多分片，起到节省带宽的目的。在实时处理的直播工作模式下，需要首先把直播流传送给 HDS，然后通过包装处理再传输到客户端，这种模式在网络直播和视频会议中被广泛应用。

（4）DASH

由于不同的公司具有不同的流媒体技术标准，且这些标准是不相兼容的，例如，苹果的 HLS、微软的平滑流媒体和 Adobe HDS 都无法相互兼容。为此，MEPG 标准工作组推出 MEPG-DASH 标准，目的在于为业界的多种动态自适应流媒体技术建立统一的技术标准，得到了许多公司的大力支持。

如图 4-13 所示，DASH 的技术架构主要由 3 部分构成。媒体内容制作和分发是由内容生成服务器完成的，DASH 内容管理并响应客户端请求是由流媒体服务器完成的，并且将媒体表示描述（Media Presentation Description，MPD）索引文件与媒体分片文件分别存储在流媒体服务器上。而 DASH 客户端，主要负责管理媒体文件下载请求、解码与输出媒体内容。如图 4-14 所示，展示了流媒体服务器与客户端间的 DASH 内容传输流程。当客户端发起媒体内容播放请求时，客户端首先向服务器请求解析 MPD 索引文件，获取节目码率等信息，然后自适应地根据实时网络带宽情况向服务器请求相应的合适码率的媒体分片文件[6]。

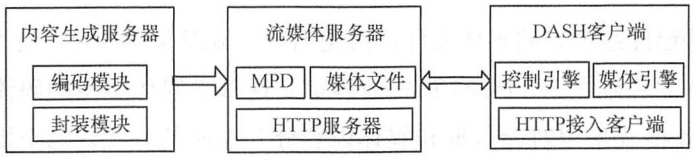


图 4-13　DASH 技术架构

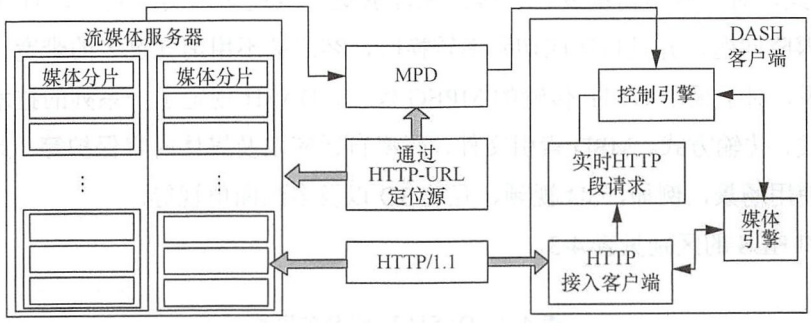


图 4-14　DASH 内容传输流程

MPD 是一种描述客户端向服务器获取媒体分片信息与解码信息的索引文件。如图 4-15 所示，从 MPD 分层数据模型结构可以看出，内容描述元数据被封装在 5 种嵌套的元素中。例如，可以在时段元素中封装广告内容，而一个节目的音频、视频、字幕等相关元数据可以分别封装在 3 个不同的自适应集或者复用在一个自适应集元素中进行描述[6]。

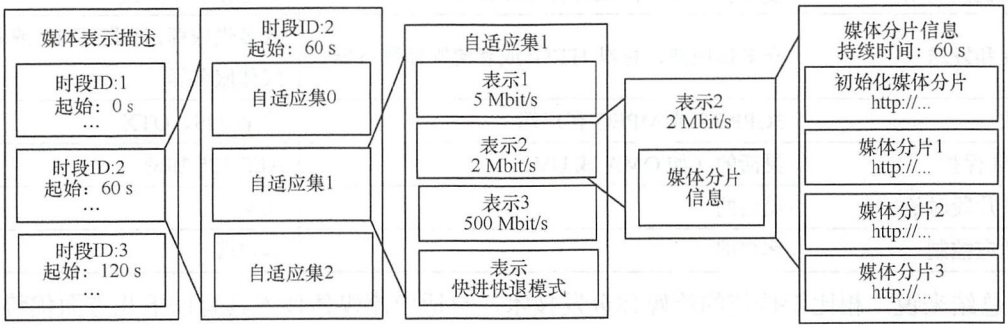


图 4-15　MPD 分层数据模型

DASH 流媒体技术支持分片 MP4（简称 fMP4）和 MPEG-2 TS（简称 M2TS）两种媒体流封装格式，fMP4 为主要流封装格式，下面对 fMP4 格式进行详细介绍。

对于同一节目，MPD 索引文件对多个音频、视频或字幕等元数据统一进行描述，服务器

不用提前存放这些已经组合好的媒体文件，而是当客户端发起请求时，会根据描述信息按需重构媒体文件。在此基础之上，fMP4 格式内容可实现基于媒体类型的码流时间同步和播放无缝切换[6]。fMP4 格式采用对元数据和媒体内容分开存储的方式，每个媒体分片包含可独立解密和解码的内容元数据[6]。fMP4 格式的媒体分片的时间严格对齐，一个分片长度可以小至 2 s，因此，对于网络情况变化，客户端能够更灵敏地做出实时响应，有效地改善了视频画面质量骤降问题，并且自适应切码率转换时，客户端不用消耗过多的带宽。

总的来说，对于采用 HTTP 传输的 MPEG 媒体，DASH 规定了一系列的技术要求，包括媒体内容格式、传输方式、MPD 索引文件、码率自适应以及媒体内容保护等，此外，还定义了一系列的使用场景，例如，3D 视频、互动 3D 以及多画面电视等。

DASH 和 HLS 的区别见表 4-3。

表 4-3　DASH 和 HLS 的区别

项目	MPEG-DASH	苹果 HLS
类型	开放的，基于标准	苹果控制
源视频编码器	H.264 等	H.264
源音频编码器	AAC 等	AAC、MP3
片段格式	MP4 格式 MPEG-2 TS	MPEG-2 TS
服务器上文件存储	连续不分片的或每个片段独立成文件	每个片段独立成文件
音频视频复用	复用在一个片段或者音频和视频分开	复用在一个片段
分片和发送	众多供应商，标准 HTTP 或者流媒体服务器	众多供应商，标准 HTTP 或者流媒体服务器
播放	3GPP R9 或 MPEG 客户端	苹果 iOS、QTX
DRM 保护	灵活的（如 OMA 或 UV）	AES-128 加密
典型片段时长	灵活的	10 s
自适应控制	客户端	客户端

总结来说，相比于传统的流媒体分发技术，自适应流媒体技术具有以下几方面优势：

- 自适应流媒体技术与传统的 HTTP 渐进式下载流媒体技术相比，最大的特点在于对直播的支持；
- 由于自适应流媒体技术使用的是通用的 HTTP，因此能够兼容传统 HTTP 缓存和防火墙等网络设备，使得网站服务器部署更加容易；
- 能够根据用户的实时带宽情况，通过自适应地动态解码来调整和优化码率，为用户带

来流畅的播放体验；

- 客户端初始化默认选择低码率，用户不需要长时间等待，能够实现快速播放。

综上所述，与传统的流媒体技术相比，自适应流媒体技术不仅能够支持直播，还能够充分利用可用带宽，提供更好的服务质量，而不是强行要求客户端选择一个低于可用带宽的固定码率播放。相信在不久的未来，自适应流媒体技术能够得到互联网视频行业的广泛认可和应用。

4.6　服务鉴权技术

4.6.1　常见的服务鉴权技术

网站的资源盗用指通过技术手段，直接在本网站上向最终用户提供其他网站的内容，从而免费获取其他网站的优质资源。资源盗用绕过了该网站通过该内容获取正当利益的页面，比如广告页，因而侵犯了网站的正常商业利益。与此同时，也增加了网站服务器的访问流量压力。因此，服务鉴权是必须考虑的问题。容易被非法盗用的内容有很多类型，比如视频、下载资源、图片等。因此，从技术角度上来说，服务鉴权也有很多种实现方式，下面介绍其中比较常见的几种。

（1）判断引用地址

判断引用地址是最早也是最常见的服务鉴权策略。判断引用地址就是通过 HTTP 头的 Referer 字段的值来获取浏览器的页面地址，这样就能知道用户此时是在本网站的页面上，还是在非法的页面上。这种方法通常用于防止图片、mp3 等被人用 HTML“嵌入”其他网站[3]。

（2）使用用户合法性验证信息

这个方法常见于用户登录认证后才能浏览的网站，比如论坛、社区、会员制网站等。当浏览器向网站资源发起请求时，网站服务器要做的第一步是确定用户是否登录验证合法，ASP.NET 中通常用会话标记登录状态，如果用户是非法的，则返回一个错误提示信息。

用户登录状态通常是由会话 ID 决定的。一般情况下，会话 ID 存储在 HTTP 请求的 cookie

字段中，也会在某些情况下放置在 URL 里，便于进行快速查询。

（3）使用 cookie 携带动态验证信息

此方法是在页面中生成动态值的 cookie，然后在处理资源请求时对 cookie 中是否有正确的值进行判断。如果没有，则返回错误消息。生成动态 cookie 值的方法有许多，只要服务器能逆向判断动态值的合法性即可。比如，对于 ASP.NET 的网页程序，直接向会话里任意存一个字符串或数字，然后在处理资源下载请求时优先检查会话里是否含有该字符串或数字[3]。

（4）使用 POST 下载

当客户端使用 HTTP GET 向服务器发起资源请求时，服务器会使用 POST 方法将数据返回给客户端。将下载链接转换成一个表单和一个提交按钮（提交），将要下载文件放置在一个隐藏的文本框里，当用户点击提交按钮时，服务程序首先确定是否用 POST 发起请求，若是 POST 请求，则读取目标资源并写入响应对象[3]。

（5）使用图形验证码

该方法需要用户输入图形验证码中的数字或者字母等，使用这个方法可以确保每次都是“人”在网站上进行下载请求操作，而不是利用下载工具获取。这个方法的缺点是比较容易让正常的用户感到麻烦[3]。

（6）使用动态密钥

当用户点击一个链接资源时，服务器首先计算一个在一定时间内有效的密钥值，然后在数据库或缓存中记录资源该密钥值和它相应的资源文件名称，最后通过特定的算法或规则生成一个包含该密钥信息的新的 URL 地址。当用户发出资源请求时，程序先检测是否有合法的密钥存在，如果存在则返回对应的资源数据。

（7）在内容中插入随机数据加密

该方法是指在文件资源中插入一些随机字节，如在 MP3 标签区、RAR 文件的备注区插入随机字节，使整个文件的散列值会发生变化，使其与别的网站下载文件的散列值是不一样的，起到加密的效果，只有通过合法性检查后，才能正常获取这些文件资源。

（8）打包下载

这个方法跟方法 7 类似，只是方法 7 是在原始文件里修改，而方法 8 是在原始的文件基础上进一步封装，同样使资源的散列值改变。但是，这种方法不适合用于在线流媒体业务，只能用于下载业务。

4.6.2　CDN 服务鉴权机制

在使用互联网应用时，往往需要用户进行鉴权，保证只有合法的用户才能使用该业务。而当网站业务由 CDN 承载后，CDN 也需能对用户进行服务鉴权。第 4.6.1 节介绍了使用应用服务的主要服务鉴权方法，其中两种分别是动态密钥和用户合法性验证。那么，CDN 是如何对动态密钥和用户合法性进行服务鉴权的呢？本节将进行简单介绍。

CDN 基于动态密钥的鉴权方式是本地鉴权。本地鉴权采用开环加解密机制，支持基于算法＋密钥的服务鉴权摘要实现鉴权，由 CDN 服务节点负责进行验证。CDN 支持采用多种不同的加解密算法和校验策略，并能够灵活配置和扩展新的加解密算法和校验策略。

本地鉴权一般采用门户和 CDN 共享密钥的机制，可使用对称算法（如 AES）进行加解密。门户在返回给用户的 URL 中增加加密后的服务鉴权信息（如防盗链字符串 authInfo），用户利用含有服务鉴权信息的 URL 向 CDN 中请求内容服务，CDN 解密后通过检查 URL，保证 CDN 服务的合法性，具体流程如图 4-16 所示。

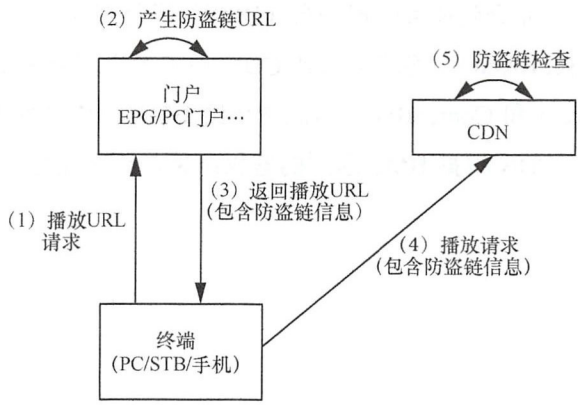


图 4-16　防盗链机制示意

开环加解密机制采用的密钥支持定期进行更新，更新时需要进行同步。前期可采用固定密钥在 CDN 和业务系统中分别进行配置，后续可考虑从业务管理系统中统一进行密钥管理下发给 CDN。在密钥同步期间，CDN 需采用合适的机制同时支持新旧密钥过渡。

CDN 利用用户登录合法性进行二次鉴权。二次鉴权采用闭环加解密机制，CDN 仅对服

务请求进行转发验证，并根据返回结果提供或拒绝服务，CDN 本身不负责进行本地验证。由业务管理系统（BMS）对用户请求的内容进行二次鉴权，流程如图 4-17 所示。

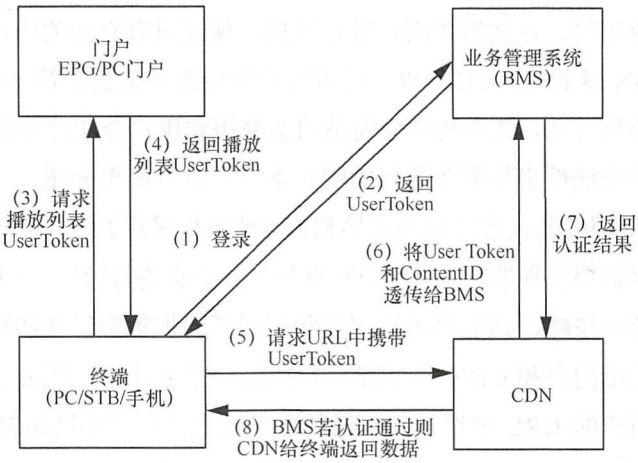


图 4-17　二次鉴权机制示意

终端每次打开时，首先会到业务管理平台（BMS）进行登录认证，认证通过后，会获取到用户合法性令牌（UserToken）。终端每次到 CDN 请求内容都会在 URL 中携带 UserToken 信息。CDN 将 UserToken 和 ContentID 等信息透传给 BMS，BMS 对此播放进行鉴权，并将鉴权结果返回给 CDN，CDN 根据 BMS 返回的鉴权结果，给终端返回媒体数据或禁止播放。

第 5 章

CDN 新技术

在第 1 章介绍过，用户在访问请求互联网应用服务时，如果超过一定的等待时间，就会放弃访问，因此，利用 CDN 加速互联网应用的需求应运而生。下面将分别介绍 CDN 对于互联网应用的几种典型加速技术。

5.1 前端优化技术

网站通常由前端和后端构成。网站业务逻辑之前的部分称为前端，包括用户使用界面、网站页面模型以及图片加载等，属于功能的呈现界面，其优劣能直接影响用户访问体验。相比之下，后端的主要作用是实现网站的功能。所以说，如果前端得以优化，能给用户带来更好的体验，如图 5-1 所示。下面介绍前端页面性能优化（Front-End Optimization，FEO）的方法，使页面更有效地呈现在浏览器中。

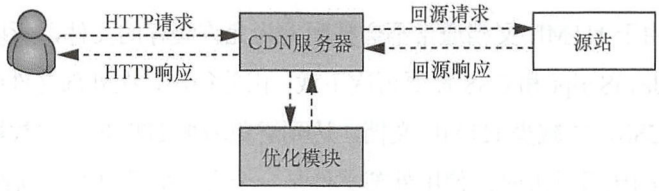


图 5-1　FEO 优化模块

（1）合并 HTTP 请求

HTTP 作为无状态的应用层协议，对于每一次 HTTP 请求，通信链路建立以及数据传输都是必不可少的，并且需要在服务器端单独启动线程去处理每个 HTTP 请求。这就代表着 DNS 查询、应用层重定向以及 404 等很可能会在每一次 HTTP 请求时发生，因此，可以通过合并 HTTP 请求的数目来有效提升访问性能，避免这些昂贵的通信服务和开销。将 CSS、JavaScript 和图片合并成一个文件是减少 HTTP 请求的主要方法，这样只需要一次请求即可。

（2）使用浏览器缓存

静态内容在网站中通常更新频率较低，如 CSS、JavaScript 以及图片等。上文说过，访问这些文件每次需要进行 HTTP 请求，那么，将这些缓存在浏览器中，则可以减少 HTTP 请求的次数，从而提升网站访问性能。浏览器缓存的设定是通过设置 HTTP 头中的 cache-control 与 expires 的属性来实现的，设定的缓存保留时间可以短到数天，长到数月。

值得注意的是，网站在利用浏览器缓存策略更新静态内容时，最好能够一个个文件逐一进行更新，例如需要更新 20 个图片时，不应该将 20 个图片一次全部集中更新，而是需要相隔一定时间间隔逐一更新。否则，用户浏览器会发生大量缓存突然失效，并且会造成服务器负载突发性加重以及网络堵塞的情况。

（3）压缩组件减少通信传输

在通信传输过程中，减少通信传输的数据量能起到减少传输负载大小的作用。通常，在服务器端对需要传输的文件进行压缩，而在用户浏览器端再对文件进行解压缩，达到压缩组件减少通信传输的目的。对于提高压缩率，可以通过将外部的脚本和样式合并为一个实现。HTML、CSS 和 JavaScript 文件利用 GZip 压缩方式使得压缩率超过 80%。然而，文件压缩对服务器和浏览器都会产生一定的压力，在通信带宽良好，但服务器资源不足的状况下需要权衡考虑。

（4）使用外部 JavaScript 和 CSS

由于 JavaScript 和 CSS 能够被浏览器缓存，因此，使用外部文件通常会较快地打开页面。而在内联的情况下，由于 HTML 文档通常不会被配置为允许缓存的文件，所以用户下一次请求内联 HTML 文档时，JavaScript 和 CSS 需要再次下载。由此得出，在外部文件中，可以通过浏览器缓存 JavaScript 和 CSS，并减少 HTML 文档，达到避免增加 HTTP 请求数量的效果。可以说，当 HTML 文档数占的比重过大时，使用外部文件是一个有效提升页面访问性能的手段。

很多时候，网站难以精确判断是否需要使用内联或者外部文件，这时使用外部文件，仍

不失为一种优良的手段。但有一个例外情况是，由于网站主页对于响应时间的要求更高，这种情况下，更倾向于使用内联文件。

（5）减少 cookie 传输

对于每一次请求和响应，cookie 过大会对数据传输造成极大影响，所以需要谨慎考虑并决定把哪些数据写入 cookie，以达到减少 cookie 中传输数据量的目的。对于访问一些静态内容，例如，对于 CSS 和 JavaScript 等发送 cookie 是没有意义的。在这种情况下，通过使用独立域名访问静态内容的方式，避免请求静态内容时发送 cookie，并减少 cookie 的传输次数。

（6）避免 CSS 表达式

CSS 表达式是一种动态设置 CSS 属性的强大且危险的方式。在人们期望中，表达式不只在页面呈现和值大小改变时需要利用表达式求值，甚至当页面上下滚动，包括鼠标在页面上移过时，表达式都要进行求值操作。使用一次性表达式能够减少 CSS 表达式求值次数。如果，CSS 表达式必须被求值一次，可以在这次中对它本身进行重写。

中国电信某测试对比结果见表 5-1，CDN 与 FEO 进行整合之后对各个页面评测指标有很大的影响，从请求数到有效负载，再到开始渲染和加载时间，都得到了很大改善，整合的 CDN+FEO 方案可以将网页速度提升 4 倍，并将总的有效负载减少 70%左右。

表 5-1　FEO 评测结果

	原始情况	利用 CDN	利用 CDN+FEO
加载时间/s	14.33	6.25	3.68
开始渲染时间/s	6.94	2.75	2.04
请求数量	77	77	10
传入请求字节/KB	924	796	424

总而言之，CDN 与 FEO 相结合，则会具有很强的互补性，可以很好地结合在一起，帮助用户最大限度地提升性能。

5.2　动态加速技术

动态内容是源站动态生成的内容，因此，更新频率较高。长期以来，对于使用 CDN 对动态内容加速的效果，一直有比较大的争议，因为在不少人的意识里，使用 CDN 无法缓存动态

内容，不缓存就不会加速。那么，CDN 动态加速到底是如何实现的？显而易见，对于动态生成的内容，用户请求到了边缘节点之后还得回源获取内容，因此，避免传输时延和数据传输失败成为 CDN 动态内容加速的主要目标。为了满足用户的动态内容访问请求并减少网络带宽和网络时延，诞生了多种动态内容分发加速技术。下面具体介绍这些 CDN 动态加速技术。

动态网页分发加速技术可以分为三大类：一是差异化缓存技术，主要包括 ESI（Edge Side Includes）规范和 CDE（Class-based Delta Encoding）[1]；二是传输加速技术，目的是减少从源服务器到用户之间传输动态内容所需的带宽和时延；三是动态内容生成加速技术，用以提高服务器端产生动态网页的性能，主要包括预取技术 DUP（Data Update Propagation）和 DC2CP（Dynamic Content Caching Protocol）[1]。

（1）差异化缓存技术

① ESI

ESI 片段缓存是加速网页应用的一种有效技术。ESI 是一种基于 XML 的标记语言，用于定义包含片段、变量和其他控制指令的模板，使得用户、代理和 CDN 能够在片段级别缓存文档。ESI 将网页分解成具有不同缓存特性的片段（可缓存和不可缓存），支持页面片段的动态装配，每个片段作为一个独立的元素保存在缓存服务器上。缓存服务器在响应用户请求时只需要获取那些不可缓存或者过期及没有命中的片段，减少了响应时间和源服务器的负担[1]。ESI 分离内容传输和内容生成如图 5-2 所示。

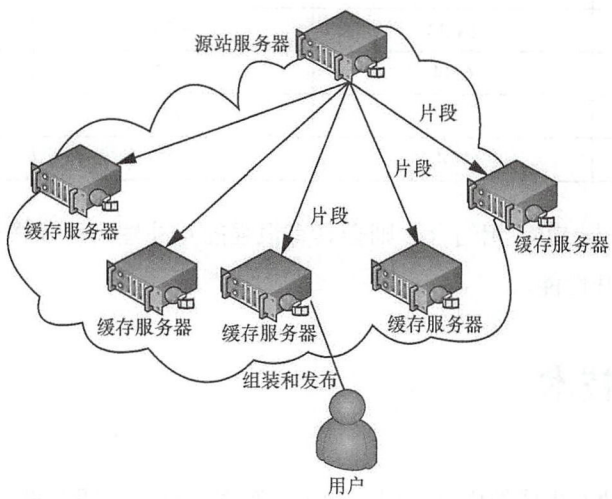


图 5-2　ESI 分离内容传输和内容生成

② CDE

DE（Delta Encoding，增量编码）指的是以基文件（定义见下文）对目标文件进行编码的过程，编码后的结果称为 Delta（增量）。该方法是在一个慢速链接的两端分别存储一个文档的相同副本，称该副本为基文件。慢速链接的服务器端从服务器取回文档的当前副本，并计算当前副本与存储副本的差异（增量），然后再把增量发送到慢速链接的另一端。当客户机收到增量以后，就把增量与基文件结合起来生成当前副本。因此，响应一个对动态网页的请求时，只需要传输编码得到增量，通过基文件和增量进行解码生成当前页面[1]。技术原理如图 5-3 所示。

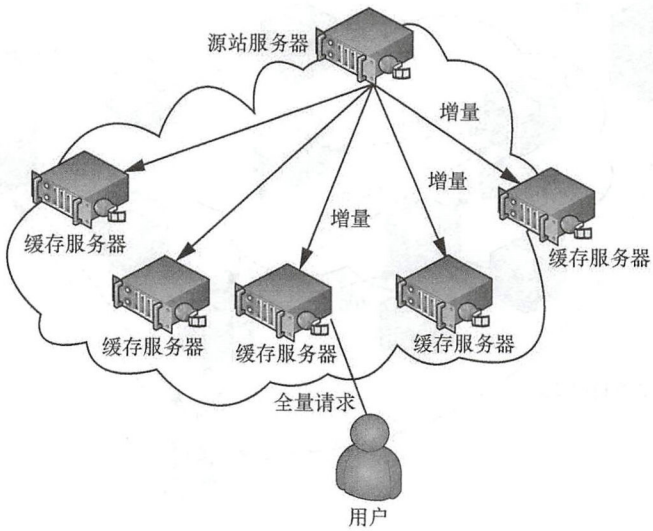


图 5-3　传输加速网络示意

DE 的缺点是不能解决服务器端的可扩展性问题，动态文档的数量在不断增加，服务器端存储容量的需求随着动态内容数量的增加而增加。对这个问题，可使用 CDE 方法[1]：

- 对多个文档进行分类，每类只保存一个基文件；
- 当服务器端对当前响应进行增量编码的时候，如果没有保存所用的基文件，则在编码后的响应中包含该基文件的 URL；
- 当用户收到响应后，如果发现没有该基文件就先请求该基文件。因此在第一次请求某网站的网页时，需要请求两次才能获得该网页。

（2）传输加速技术

对于动态生成的内容，用户请求到了边缘节点之后需要回源拉取内容，因此，避免传输时延和数据传输失败并以最快的路径传送至对端，是 CDN 动态内容加速的主要目标。

动态内容传输加速的核心是动态路由优化技术，CDN 的全局调度内容路由 RR 首先将用户请求引导到距离其最近的边缘节点，通过该边缘节点进入 CDN。然后，通过 CDN 节点众多的优势，把每个节点都视为一个路由，通过实时采集网络链路质量、节点部署位置、跳数、往返时间以及分组丢失率等信息，在用户进入的边缘节点和源之间寻找到传输质量最佳的路径，如图 5-4 所示。

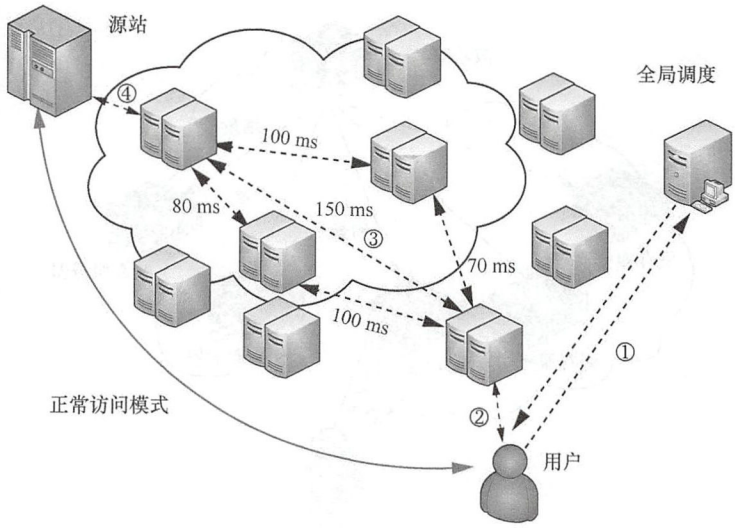


图 5-4　传输路径示意

由于 CDN 内部节点之间的专有链路传输质量好、带宽足够，因此，服务质量通常能够得到有效保障，然而，为进一步提升动态内容传输速度，还会在传输协议上进行改进，如下所示。

- 连接复用：通过节点与节点、节点与源站之间的 TCP 连接复用，保证每次动态请求到达时，服务节点和源站之间的通路中连接都已经建立通路，减少因 TCP 建立连接所引发的时延，但在突发情况下很难保证。
- TCP 优化：定制和参数优化 TCP 的协议栈，调优传统 TCP 的拥塞控制机制、慢启动

机制等。

- 私有传输协议：采用私有的传输协议代替 TCP，进一步提升 CDN 内部内容的传输速度。

（3）内容生成加速技术

① 预缓存

预缓存是根据用户当前的访问请求和历史访问记录，预测用户将来可能发出的访问请求，在用户浏览当前网页时将预测的内容存储到本地高速缓存中。用户在真正要访问这些页面时只需从本地高速缓存下载，从而达到快速响应的目的。预取也被称为主动缓存技术，主动缓存方案（Active Cache Scheme）通过一种 CacheApplet（CacheApplet 是由源服务器提供给代理的代码，可以通过其 URL 获取）的方式把部分用户请求处理的任务转移到代理上。当缓存网页时，代理会从源服务器上获取相应的 CacheApplet。当用户请求命中缓存的动态网页时，代理把用户请求作为参数传递给已下载到本地的 Applet，Applet 根据情形选择：通过计算返回一个新的页面、返回对象的副本或者要求代理转发请求到源服务器[1]。

② DUP

DUP（Data Update Propagation）可以确定底层数据的改变对缓存对象产生的影响。它通过建立底层数据和缓存内容的精确依赖关系，并计算和实时更新这种依赖关系，触发缓存内容更新。当底层数据变化的时候，系统更新或者使缓存对象失效。缓存内容可能是整个网页，也可能是网页中的片段。当系统感知到底层数据改变时，就使用算法确定受到影响的缓存对象。一旦发现缓存对象过期就让它失效或者进行更新[1]。

③ DCCP

DCCP（Dynamic Content Caching Protocol）支持个性化内容生成程序（如 CGI 应用软件、Javaservlets 等）指定不同 GET 请求之间的等价或者部分等价，网页缓存可以依据该信息来响应相同或者等价的请求。对于部分等价的请求，当生成和传送实际内容的时候，可以把已经缓存的内容作为一个近似的结果进行响应[1]。

上文介绍了几种典型动态内容分发加速技术，整体可分为差异化缓存技术、传输加速技术和内容生成加速技术。对于差异化缓存技术，CDE 可以有效地减少响应时间和带宽需求，但是个性化长尾效应越明显，该技术获得的效果就越不显著；ESI 有效地重复利用了片段性内容，但是产生了对内容进行分片的开销。对于传输加速技术，主要以动态路由为核心，降低回源获取内容的传输时延。对于内容生成加速技术，比较容易在服务器端用增加硬件和部署新技术的方式来提高动态内容的生成速度。

5.3 SDN 调度技术

SDN 作为一种新型网络设计思路，控制平面和数据转发平面分离，不再紧耦合在一起，是 SDN 与传统网络设计方式的根本不同之处，如图 5-5 所示。

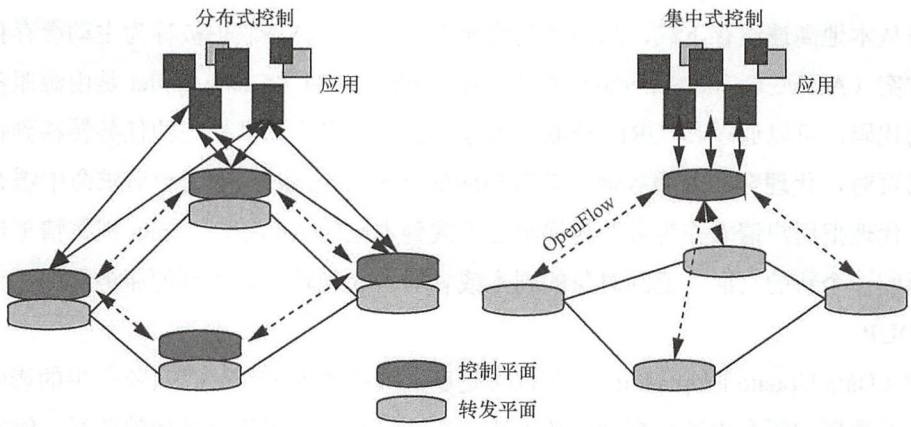


图 5-5 传统网络架构与 SDN 架构

SDN 和传统网络相比，主要具有以下两方面优点：

- 整个网络的状态信息都是由 SDN 控制器收集的，逻辑上集中的控制平面能够控制多个转发面设备，并且为了实现负载均衡，通过全局最优的方式调控网络流量和动态路由；
- 网络管理控制是通过软件编程来实现的，不仅灵活配置，而且无需关注底层实现细节。

由于开放能力不足，传统 CDN 难以通过感知网络状态来定义转发策略，也无法实现动态的底层资源跨区域调度，并且一般的优化策略难以显著提高性能。因此，需要软件定义网络（SDN）这种新体系架构为 CDN 提供整体支撑能力。如图 5-6 所示，物理基础设施平面、内容资源管控平面、内容资源调度平面和业务应用平面组成了基于 SDN 的 CDN 部署架构。

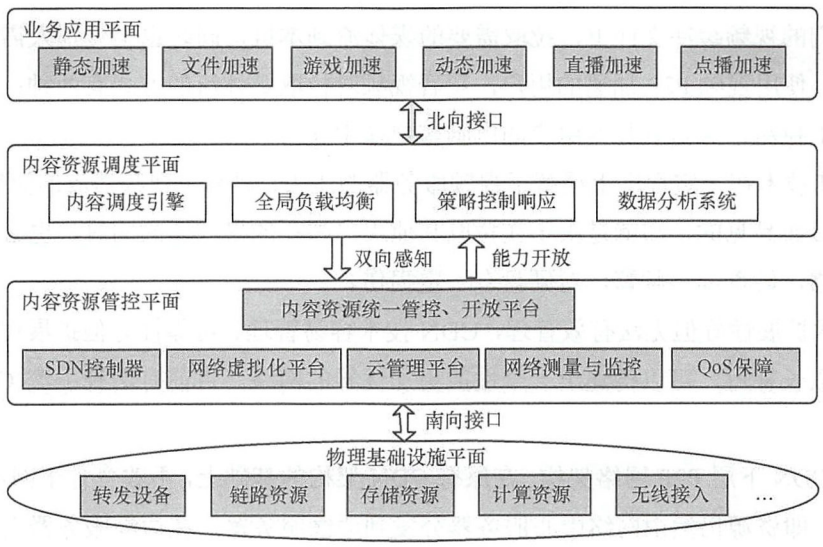


图 5-6　基于 SDN 的内容分发系统架构

对于基于 SDN 的 CDN，用新内容资源调度平面替代原有的全局负载均衡调度系统，并通过管控平面提供的数据实现调度，控制资源分配，实现全局负载均衡。此外，通过 SDN 的管控平面，CDN 在获得更详细的网络状态数据时，不仅能够实现精细粒度的策略控制，并能够基于用户资源请求进行大数据挖掘。管控平面上层为业务应用平面，用于提供不同类型内容的加速服务。由此，基于 SDN 的 CDN 系统实现了控制与转发分离以及功能与实现分离，同时提高了 CDN 加速服务的扩展性。

5.4　P2P 流媒体技术

总体来说，通过管控平台获取核心数据，资源调度平面使业务应用平面与内容资源管控平面高效交互，是基于 SDN 部署 CDN 系统提升系统性能的关键之处。借助 SDN，灵活地实现虚拟网络划分机制，充分利用网络资源，不必再担心因为 CDN 底层资源不足而造成性能下降，从而实现全网级流量优化。P2P 技术也是在流媒体直播业务的催生下红极一时，用来提高视频直播的服务质量。P2P 技术对直播视频加速是利用视频网站或在线播放器提供的视频 P2P 加速插件实现的。

P2P 插件会自动检测同一播放器中观看同一视频，且同时使用了视频加速功能的电脑客

户端，从它们的视频缓冲文件中，拉取需要的媒体流到本机，同时也上传别人需要的数据到网络。因此，使用视频 P2P 加速的用户，观看视频时拉取媒体流的方式有两种：一是直接从视频服务器上拉流，二是从其他用户的电脑客户端中获得。

P2P 加速技术在一定程度上缓解了视频服务器的压力，提高了视频播放的流畅度。然而，其缺点也是显而易见的，那就是由于在拉取其他用户缓存的媒体流的同时，也必须上传相应的数据到网络，硬盘读写频繁，对硬盘有一定损伤。

P2P 技术扩展性好但无法有效管理，CDN 技术容易管理、可靠性好但扩展性差，将两种技术融合，取长补短，可以构建出一个性能更加优异的网络。目前有两种主流 CDN 和 P2P 融合架构。

- 上层 CDN 下层 P2P 网络架构。在原有 CDN 架构的基础上，不改变骨干网层次的 CDN 结构，即资源仍然由网络中心服务器分发到边缘服务器，在边缘服务器引入可管理的 P2P 自治区。自治区由单个或多个边缘服务器和其网络覆盖下的多个用户构成，自治区下用户之间使用 P2P 技术传输资源，有效地减少骨干网的压力。
- 在边缘服务器间建立 P2P 网络的架构。这种架构下不需要引入 P2P 自治区，在原有的 CDN 基础上，以 P2P 架构的方式将边缘服务器组成起来，实现边缘服务器之间的内容交换，大大降低了中心节点的负载。因此，CDN+P2P 融合架构流媒体对直播服务具有良好的加速作用。

5.5 应用协议加速技术

应用协议加速是通过对 TCP/IP 等传输协议的优化，改善和加快用户在广域网上的内容传输速度，或者对一些特定协议（如 SSL 协议）进行加速，解决安全传输时的性能和响应速度问题。主要的应用协议加速服务有以下几种。

（1）广域网应用加速

其目的是处理多种分布式企业网络环境下的各种应用和协议，尤其针对那些在局域网上可正常运行，在广域网却受到极大影响的应用和协议，比如 CIFS 协议、NFS 协议等。广域网应用加速能够在不改变远程用户使用习惯的前提下，将分布式的 IT 基础设施如文件服务器、邮件服务器、网络附加存储（NAS）和远程办公室备份系统等集中起来，整合到统一的

数据中心。此外，广域网应用加速还支持通过长距离广域网链路进行文件备份与复制操作，在不升级带宽的前提下在现有的广域网上提供比以前丰富得多的应用服务[1]。

（2）SSL 应用加速

其目的是在保证安全性的前提下提高 SSL 应用的响应速度，有效减少源站点服务器在 SSL 应用方面的资源开销。目前许多基于网络的重要核心应用都采用了 SSL 技术来保证服务的安全性和私密性，但是由于 SSL 应用需要进行大量的加密/解密运算，其对服务器端的资源消耗是非常巨大的。CDN 提供 SSL 应用加速后，由 CDN 的专用 SSL 加速硬件来完成加密/解密运算工作，通过认证之后建立起数据传输通道，用户的源站点无需面对海量用户，只需信任有限的 CDN 缓存即可，从而减轻了运算和认证的资源开销。该应用适用于进行网络办公的企业和电子商务网站及金融网站，如分支机构遍布全球的大型企业、B2C 的网络交易平台等[1]。

（3）HTTPS 加速

随着 HTTPS 应用越来越广，CDN 承载 HTTPS 的需求越来越大。HTTPS 加速（HTTPS Accelerator）是拆分 HTTPS 请求，通过 HTTPS 加/解密优化、静态资源边缘缓存、动态数据全网加速、动静混合数据自动分离等技术实现 HTTPS 加速传输；HTTPS 加/解密优化方案包括握手协议优化、证书验证优化；静态数据边缘缓存方案包括层级收敛、定制缓存和分层加载；动态数据全网加速方案包括智能路由、传输协议优化和差分传输；动静混合数据自动分离方案包括动静态数据自动分离后静态数据边缘缓存、动态数据全网加速。

5.6　智能协同技术

CDN 中存放着大量的业务使用数据，利用这些业务数据并加以内容管理策略辅助，为 CDN 中内容的智能存储、智能调度以及智能组网提供了重要基础。

（1）智能存储

传统模式下，CDN 是通过整片完全存储的方式存储片源的。假如一共有 1 000 部片源，那么，边缘节点按照整片完全存储其中 20%，也就是 200 部，各个区域节点（也被称为二级节点）存储其中 80%，也就是 800 部，对于中心内容库节点，为了下级节点能够回源调用内容，通常会完全存储全部的片源。随着用户对视频业务需求量和视频画面

清晰度要求的增长，采用整片完全存储时，平均每部片源的大小和需要存储的片源数量都会急剧增长，这将需要足够大的存储空间，并且根据传统整片存储方式组织的 CDN 存储将带来高昂成本。自此，CDN 存储开始以文件切片方式为基础进行发展。该方法将一个完整片源切成许多个切片，每个切片依据用户访问的热度信息分散存储在不同节点上，不必重复存储完整片源。对于 CDN 的智能存储，分片的热度情况判断以及跨节点的内容快速调度是最关键的。

（2）智能调度

一直以来，IP 地址就近性、服务器负载状况以及内容命中情况是 CDN 调度的主要依据。但随着新业务的发展，用户使用情况开始被要求作为 CDN 调度的依据，为 CDN 智能调度奠定基础。用户使用情况包括用户的身份验证、用户使用行为数据以及用户所在地区网络情况的判断。具备这些用户使用情况的数据后，加以分析利用，才能使业务平台智能地提供应用服务，例如连续播放跨屏内容、用户行为偏好推荐以及适配不同接入带宽等服务。以用户为中心的信息数据库的建立是智能 CDN 调度的关键点。对于视频业务，哪些片源被访问的次数多、某个片源的哪一部分访问量大以及片源被访问的时间和区域等相关信息对智能地提供业务起着重要作用。借助外部认证系统，CDN 获取用户信息，并围绕用户将业务使用环境与使用记录进行关联分析形成调度策略。

（3）智能组网

前面介绍过，由于业务类别不同，对 CDN 服务器的要求也就会存在差异，并且对 CDN 的技术能力要求也各不相同。对于视频业务，要求 CDN 能够实现大文件高缓存，并对流服务传输质量进行优化；而网页加速业务，要求 CDN 能够实现小文件高吞吐。所以说，业务特性的差异，就会导致 CDN 服务器硬件和软件配置也各不一样。因此，智能组网的 CDN 在统一架构和统一管理下，根据业务的不同进行垂直方向组网，即统一设置管理、全局调度系统以及标准化接口。对于不同业务，可采用不同技术体制的方案，各业务能力之间水平方向不进行互通。这样一来，当新业务能力需要扩展时，只要增加新的垂直业务子网即可，完全不会对已有业务产生影响。

现在，互联网行业已经大规模地商用虚拟组网技术，CDN 目前也逐步尝试采用虚拟技术组网。在统一的虚拟资源池管理中，业务关闭时同步释放虚拟机中的资源。这一方式，不仅能够大大提高业务上线速度，更极大程度地降低了业务试错成本，非常适合在目前快速发展的互联网生态圈下大力发展。

5.7　NVMe 技术

我们知道，CDN 服务器对硬盘 I/O 的要求非常高，特别是在视频 CDN 方面，要求硬盘能进行快速的写入与读取。因此，无论 CDN 服务器硬盘是从原来的 SATA 向 SAS，还是再向 SSD 硬盘不断演进，目的都是加快硬盘 I/O。

由于成本的原因，现有 CDN 服务器往往使用分级缓存的模式，即分为内存>SSD>机械硬盘的分级缓存，例如，在直播缓存中，在软件上把最热的内容放在内存，次热内容放在 SSD，冷片放在机械硬盘。随着分级缓存技术的成熟，SSD 存储的 I/O 也显得越来越不能满足业务的需求，因此引入新的存储技术的需求也越来越急迫，NVMe（Non-Volatile Memory express，NVMe）技术应运而生。

NVMe 作为一种计算机软件与非易失性存储器子系统通信的接口，针对企业和客户端固态驱动器进行了优化，通常作为 PCIe 接口的注册级接口。

围绕 NVMe 展开的技术工作在 2009 年正式启动，在 Intel 作为主要牵头人的带领下，90 多家公司组成的工作小组制定了 NVMe 技术规范，其他制定规范的成员公司包括美光、戴尔、三星、Marvell、NetAPP、EMC 以及 IDT 等。NVMe 技术规范的制定是为了建立新的 SSD 存储规范标准，使其从陈旧的 SATA 与 AHCI 中脱离出来。NVMe 标准在 2011 年正式问世，该标准是基于闪存存储的特点而定制的，并且解除了旧标准施加在 SSD 上的种种限制条款。2012 年，NVMe 技术升级到 NVMe 1.1 版本，截至目前，NVMe 标准已经发布了 1.3 版本。与 AHCI 一样，NVMe 也是逻辑设备接口标准，是使用 PCIe 通道的一种 SSD 规范。在刚开始设计 NVMe 的时候，就充分利用 PCIe SSD 的低时延和并行性以及当代处理器、平台与应用之间的并行性，为主机的硬件与软件提供作用。和 AHCI 标准相比，NVMe 标准能够对多方面的性能提高起到关键作用。

NVMe 相对于普通 SSD 硬盘，其优势包括：

- 性能提升数倍，读写性能是目前主流 SSD 硬盘的 3～5 倍；
- 精简了协议层，时延大幅度降低；
- 大大降低了自动功耗状态切换与动态能耗管理功能的功耗；
- 驱动适用性广，不需要厂商提供对应的驱动程序就能够正常工作；

- 能够方便适配不同平台和系统，目前，NVMe SSD 能够支持 Windows、Linux、Solaris、UNIX、VMware 以及 UEFI 等。

因此，在 CDN 服务器中引入 NVMe 硬盘将可大大提升 CDN 的服务性能，但也应看到，由于 NVMe 目前成本还较高，因而在使用 NVMe 硬盘的 CDN 中，应加强热点分析的准确性，只有精确实现热点内容的分级缓存，才能最大限度地发挥 NVMe CDN 的整体性能优势。

第二部分

CDN的选择

第 6 章

CDN 产业与市场发展

6.1 CDN 产业发展

在互联网技术发展的过去十多年间，CDN 技术创新起到了极大的促进作用，主要体现在减少网络拥塞、提高用户体验和服务质量、更有效合理地分配资源这 3 个方面，特别在网站加速方面为互联网应用发展起到了很好的推动作用。

6.1.1 CDN 产业的发展历程

1998 年，一位计算机科学学者 Tim Berners Lee 与一位数学教授 Tom Leighton 邂逅在 MIT（美国麻省理工学院）的大楼旁。作为万维网的创始人，Tim 发现访问网站变得非常拥堵和缓慢，因此，向 Tom 教授请教优化网站访问速度的算法。后来，Tom 教授和他的学生 Danny Lewin 不仅编写了该算法，还因此取得了世界瞩目的成就，并书写了 CDN 行业的里程碑——诞生了世界上第一个 CDN 服务商 Akamai。

CDN 技术的发展立足于互联网应用的发展基础上，伴随着互联网的发展而不断更新和成长。

CDN 行业的第一次危机发生在 2001 年。第一次互联网泡沫破碎，导致大量公司倒闭、

网站停止运营，对 CDN 的需求也随之骤减，包括 Akamai 公司在内的整个 CDN 行业进入发展停滞期。

从 2002 年起，宽带技术逐渐发展，并开始在全世界慢慢普及，使用户的接入带宽提高到 Mbit/s 级别，为网络提供流媒体服务创造了基础条件。2004 年以后，互联网技术发展开始回暖，下载业务产生的数据量极速飞升，流媒体业务开始兴起，网络游戏行业也逐渐成熟，这些对服务器和网络带宽有着非常大的压力，人们对 CDN 技术的需求又开始回升并持续增加，服务加速的市场需求十分迫切。

从 2005 年以后，随着视频业务的普及，我国 CDN 市场迅速升温，进入快速的阶段，其中，网宿、蓝汛为国内 CDN 服务商的典型代表，但与国外 CDN 运营的水平、技术服务能力以及市场成熟度相比，还存在一定程度的差距。直到 2008 年，伴随着我国互联网行业的大幅度升温，互联网公司、电信运营商也开始纷纷介入 CDN 市场，企图争得 CDN 行业的一席之地。2009—2010 年，网宿、蓝汛两家 CDN 服务商正式上市，我国 CDN 市场营业额达到 5.01 亿元，由此，中国 CDN 市场正式走向成熟期。

随着经济贸易的全球化发展，国外的 CDN 服务商也进入中国市场，加剧了国内的 CDN 服务商的市场竞争压力。这就要求 CDN 在技术创新和服务创新上大力发展，才能成功应对激烈的 CDN 市场竞争。

近些年，网站的内容类型越来越丰富，CDN 提供的服务也从单纯的内容加速拓展到应用和服务的加速。为了满足下载、流媒体视频、社交网络等一系列互联网服务的加速需求，在传统 CDN 技术的基础上，CDN 服务又加入了数据压缩、流量整型、应用协议加速、智能路由等网络优化新技术。

2011 年，云计算的热潮席卷而来，很多 CDN 服务商开始在 CDN 系统中应用云计算技术，这对 CDN 行业的发展也起到了不小的作用。通常情况下，云计算平台在对外提供服务时需要用到 CDN 的分发能力。同时，CDN 的技术特点本身就类似于一种云服务，这样看来，两者似乎本来就应该结合是起来加以利用。云计算与 CDN 技术的结合是一种良性的促进，推动了 CDN 技术和业务发展迈向一个崭新的阶段。

2015 年，“宽带中国”专项行动落地，国内“互联网+”计划也逐步推进，CDN 作为网络基础设施，进入了更快的发展阶段，发挥巨大作用。目前，CDN 市场需求旺盛，无论是第三方 CDN 服务提供商还是自建 CDN 服务，都会引发业界的广泛关注。

6.1.2　CDN 服务提供商类型

CDN 市场迅速发展，涌现出了一批 CDN 服务提供商。目前，国内和国外 CDN 服务提供商主要包括 4 种类型：第一类主要是包括蓝汛、网宿、帝联以及 Akamai 在内的传统 CDN 服务提供商，通过自研 CDN 系统，为网站和企业提供专业 CDN 服务；第二类是包括阿里、腾讯以及亚马逊在内的基于云计算的 CDN 服务提供商，利用云计算平台能力为用户提供一体化的 CDN 服务；第三类是包含中国电信、Level 3 在内的电信运营商；第四类是包含网心科技、七牛在内的业务专注型新锐 CDN 服务提供商。这 4 种不同类型 CDN 服务提供商对比见表 6-1。

表 6-1　不同类型 CDN 服务提供商对比

CDN 服务提供商	优势	劣势
传统 CDN 服务提供商，以网宿、蓝汛、帝联、Akamai 为代表	CDN 服务提供商的鼻祖，拥有大量成熟的 CDN 运营经验和服务能力，有利于扩大市场规模	必须消耗成本费用租用电信运营商带宽，且带宽计费方式固定，不能按需索取，因此，CDN 整体定价会比较高
基于云拓展的 CDN 服务提供商，以阿里、腾讯、亚马逊为代表	(1)积累大量云服务用户，基于云服务拓展，且容易获得用户行为大数据，具有巨大的商业价值； (2) 依靠互联网巨头的雄厚技术和资金，有实力加大资源投入，能够在传输、存储、计算和安全 4 个方向进一步丰富与强化平台整体服务能力； (3) 带宽复用冗余大，节点分布众多，具有技术平台完备的属性	CDN 目前仅是云服务提供商的一项业务分支，因此，很多时候增值业务的丰富程度、价格与稳定性不如传统 CDN 服务提供商
电信运营商 CDN 服务提供商，以中国电信、Level 3 为代表	(1) 具备自身带宽和网络优势，能够按需所取，灵活配置 CDN 的承载能力； (2) 具备品牌效应，许多互联网公司与之密切合作，对 CDN 行业的发展方向起到引领作用； (3) 已经建成许多网络基础设施，CDN 能够在此基础上建设和升级	(1) 电信运营商技术力量储备不足，且缺乏 CDN 运营经验，可能需要通过与互联网公司和传统 CDN 服务商合作来完成 CDN 系统的开发维护； (2) 不同电信运营商之间的互联互通问题比较棘手，难以实现跨电信运营商网络的 CDN 服务
业务专注型新锐 CDN 服务提供商，以网心科技、七牛为代表	业务专注型，有创新能力，往往在细分行业中以技术挑战传统通用 CDN 模式	业务过于专注，CDN 通用服务能力弱、稳定性差

目前，互联网领域的网络游戏、支付电商以及流媒体视频等在高速发展，专业的 CDN

服务提供商需要具备完整的 CDN 架构和高效的运作调整能力，快速应对互联网应用的需求，并通过资源配置提供合理的 CDN 服务。

运营商和互联网公司的进入，可以促进整个 CDN 产业的发展，比如，阿里云进入 CDN 行业，可以让 CDN 市场格局变大。由于阿里平台拥有大量的客户群作为基础，未来这些客户群体很有可能成为 CDN 的大客户或者目标客户，并为这些客户提供标准的 CDN 服务，但专业的第三方 CDN 服务公司会为中大型客户提供更加定制化的针对性 CDN 服务，因此，市场上需要独立的外包 CDN 服务提供商。并且相对于电信运营商或互联网服务提供商，从服务专业性和服务能力的角度上看，专业的 CDN 服务商会更有优势。

当电信运营商作为 CDN 服务提供商时，具有网络资源全面部署的优势，但是由于 CDN 技术储备薄弱以及 CDN 运营经验的缺乏，单一的电信运营商很难提供跨电信运营商网络的 CDN 服务，这些都是电信运营商开展 CDN 服务最需要击破的关键点。

6.1.3 CDN 市场的发展特点

随着 CDN 技术的革新，从传统的 CDN 服务商到现在的运营商、互联网公司都纷纷介入 CDN 行业，打破了原始的 CDN 市场格局。现如今，CDN 服务市场已经成为互联网技术市场的潮流领域，其市场的发展特点主要体现在以下两点。

（1）行业发展迅速，行业利润持续增长

从 2006 年开始，CDN 厂商之间产生了激烈竞争，CDN 市场在厂商竞争中大幅度扩大。根据多家 CDN 服务提供商 2015 年的年度财务报告，借助互联网发展的良好势头，CDN 行业高速稳步发展。特别是 Akamai 公司在其财务报告中提出，依靠互联网行业的势头，CDN 业绩上升 10 多个百分点已是常态。而在国内，以网宿公司为例，在 2015 年，其 CDN 行业净利润同比增长 72.1%，高达 8.3 亿元，充分说明了现阶段 CDN 行业的利润是十分可观的。

（2）用户对 CDN 服务提供商需求大幅增长

在第 1 章介绍过，用户访问体验、网站响应速度与可靠性已成为互联网应用吸引用户的关键因素。近几年，随着海量动态和丰富的内容植入互联网，人们对移动互联网的需求显著提高，其中，电子商务、网络游戏以及流媒体视频等内容服务构成了最主要的市场增长助推剂。特别是高清视频的普及，更大程度地刺激 CDN 进一步加速服务的需求。这就要求 CDN

服务将利用技术创新减少移动互联网环境下的网络拥塞，快速响应用户的访问请求，并提高 CDN 服务的稳定性。此外，随着行业认可度和 CDN 对未来其他产业作用的提升，CDN 的社会需求性也将随之逐步提高。

6.2　CDN 发展趋势

CDN 行业见证了中国互联网的飞速发展，并且也与互联网的发展一脉相承，“互联网+”的环境模式毫无疑问成为了 CDN 服务滋养的优质土壤。在需求方面，视频直播、游戏等快速增长也为 CDN 发展提供了持续动力，同时移动互联网、云计算、大数据、物联网等新兴技术的快速成熟也引发了 CDN 技术创新的浪潮，随着 5G 的兴起，更为 CDN 的发展提供了新动力，CDN 的技术方向、市场格局、标准策略均面临深度调整。此外，由于内容入口的竞争加剧，网站攻击、劫持与内容篡改等安全事件发生的频率越来越高，CDN 平台的防攻击及 IDC、云的一体化防攻击措施将会大力发展。

6.2.1　CDN 业务发展趋势

预计在 5~10 年时间里，CDN 行业将形成包括专业技术发展和内容分发服务在内的产品链，同时，随着互联网业务的不断创新，新的服务和应用不断涌现，新兴的增值服务和新业务将会在内容分发领域得以应用。

（1）视频成为最主要业务

在国内市场，目前快速增长的应用典型代表是视频业务，如抖音、快手、微视、斗鱼、花椒、映客等短视频或直播应用。视频业务市场的火热，给 CDN 赋予了很多新的定义，也成为 CDN 产业发展的助推器。相关数据显示，2015 年中国在线视频行业市场规模达到 246 亿元，并预计 2018 年市场规模将会大幅度增长，从而推进 CDN 行业规模的扩大。

另外，互联网数据流量业务的公开数据统计显示，目前国内市场，42%的业务是视频，37%是下载业务，图片和网页文件业务加起来仅占 21%。并且，中国在线视频行业的规模分析中也表明了，2014 年在线视频的规模增长达到了 80.30%，而 2015 年也有 50.20%的增长。思科在全球范围内的研究表明，视频正在并且将继续成为促进互联网流量增长的“中坚力量”。到 2019 年年底，视频流量将占全球互联网流量的 79%，成为互联网流量增长的最直接

因素。上述各种数据都说明了视频在互联网市场正处在高速发展的时期，并且成为 CDN 业务快速发展的主要推手，如图 6-1 所示。

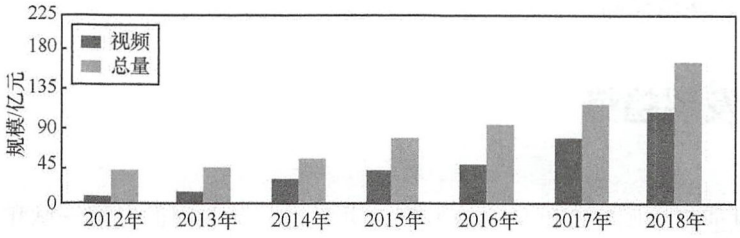


图 6-1　视频成为 CDN 发展的主要推手

在直播业务这一新兴技术为导向的互联网模式面前，传统 CDN 不足之处表现得十分明显。由于不了解细分行业的业务，并且缺乏核心技术作为支撑，为提供针对性服务造成了巨大阻碍。

事实上，视频直播行业正呈现出专业化发展的趋势，用户希望 CDN 服务提供商不仅在内容分发技术上有所建树，在编码优化、体验优化以及节省带宽的基础上，更能对其他多样化功能有所创新，提供更多的产品和服务。所以，用户对视频体验满意度成为了推动未来 CDN 产业发展的一大动力。CDN 服务提供商越早渗透视频直播业务，就会在 CDN 行业发展中领先一步。

（2）增值业务快速增长

目前 CDN 行业正经历一个重要拐点，随着亚马逊、阿里、淘宝等互联网公司的加入，CDN 行业的竞争将不仅仅局限在 CDN 领域，各种 IDC+CDN、云+CDN、安全+CDN 等应用场景催生，包括存储、转码在内的增值业务领域将会持续发展。

另外，一些动态实时交互性应用，如在游戏领域的卡牌类、王者荣耀等风靡开来，这类游戏对 CDN 应用加速同样有着紧迫的需求。

此外，在一些电商营销领域，可能存在竞争对手之间发动 DDoS 攻击，从而实现恶意攻击的目的，这就对 CDN 服务保证网站的安全性提出了要求。有些 CDN 客户希望网站能够具备阻挡 CC 攻击和致命的大流量攻击，分散目标流量，防御 DDoS 攻击的能力。

随着 HTTPS 的应用越来越广，CDN 承载 HTTPS 的需求也越来越大。一些客户希望能够实现 HTTPS 加速，拆分 HTTPS 请求，通过 HTTPS 加/解密优化、静态资源边缘缓存、动

态数据全网加速、动静混合数据自动分离等技术达到 HTTPS 应用加速的目标。

由于以上各类增值业务需求日益迫切，CDN 服务提供商只有更加深入理解和支持客户细分需求，才会在竞争中更有价值并且使整个 CDN 行业格局发生颠覆性发展转变。

（3）差异化定制需求增长

随着互联网新应用的不断涌现，互联网服务竞争的加剧，互联网应对用户差异化服务需求、对用户体验极致化追求更加明显，需要 CDN 服务提供商不仅仅能够提供文字、图片、视频、网游等应用加速，而且要求 CDN 服务提供商能够定制极致性能的独特解决方案。

例如，直播是当前互联网最热门的领域之一，各大直播类应用万花齐放，竞争激烈，只有把握好风向标，推出差异化功能服务，才能在市场中站稳脚跟，获得不可动摇的地位。因此，越来越多的直播形态出现，除了直播，还有丰富的用户交互，比如弹幕、对话等。而这些场景对于时延的要求更为苛刻，传统专注于 TCP 的解决方案往往不能满足要求。因为不仅 TCP 建立的繁琐过程增加了业务响应时间，其分组丢失重传及流控机制也难以适应直播业务实时性响应的要求。

为此，有些直播 CDN 通过私有 UDP 传送直播内容实现差异化功能，比如 YY、映客直播等。基于 UDP 的私有协议是面向无连接的，十分适合应用在实时音视频系统中，避免了 TCP 做网络质量控制所需要的开销，在网络环境较差的情况下，通过一些定制化的调优可以达到比较好的优化效果，降低直播时延。在网络环境比较恶劣或者在跨国互通的情况下，使用基于 UDP 的私有协议，时延可以小于 1 s，甚至小于 500 ms。

这样一来，直播既拥有超低时延的优势，又保留了标准协议普遍被 CDN 支持的好处。

互联网应用发展日新月异，只有不断革新技术，提高产品的性能与技术含量，从而可以实现 CDN 服务的差异化、个性化的特色，才能在 CDN 行业中保持竞争优势。

6.2.2　CDN 市场发展趋势

（1）行业竞争加大，资费逐年降低

任何行业的市场竞争手段都是多种多样的，但降价一直是最重要的手段之一。由于竞争激烈，CDN 服务的定价环境发生了变化，资费连年下降。在美国纽约召开的 2017 年 Content Delivery Summit 大会上发表了针对全球 CDN 行业发展的研究性报告，商用 CDN 的价格在

2015 年整个降幅在 20%，而最大降幅几乎达到了 45%，在 2016 年平均降价 22%，CDN 服务市场单价大幅下降。特别是大量新进入 CDN 行业的 CDN 服务提供商，为了获得市场份额、赢得客户，采用产品低价的竞争手段。并且传统的 CDN 服务提供商也加入了这场抢占市场的价格战。

尽管 CDN 行业整体价格已出现大幅度下调，而且部分企业降价幅度高达 30%。对于 CDN 业务的用户们来说，这是个好消息。但对整个 CDN 行业而言，当价格降低到一定阈值的时候，继续竞争下去永远都不会有利可图，因为风险投资对于回报能力和时间是有要求的。

未来这个行业需要改变的，远不止降价那么简单。CDN 服务的价格下降，进一步推动了 CDN 行业在服务和技术上争取用户的发展策略，并带来了销售导向、技术发展和客户利益等这些更深层次的问题。

CDN 市场会通过自我调节实现竞争性的平衡，服务使用者会逐渐回归理性，逐渐认识到服务的价值，认可高品质服务的价格。从这个角度讲，谁能够获得更多的客户，就能够在未来享有 CDN 服务的重新定价权，进而在 CDN 行业竞争中取得领先地位。

（2）客户逐步自建，自建和租用结合

对于 CDN 服务的使用，主要存在自建 CDN 和外包第三方 CDN 两种方式。一般来说，互联网企业将会在业务刚开始发展时选择租用 CDN。但是，随着互联网业务的持续发展，带宽流量在大幅度提升，业务量达到一定规模，单纯的第三方 CDN 服务往往不能满足互联网企业的要求，企业纷纷开始自建 CDN，优化用户极致体验，并选择部分流量外包给 CDN 服务器，以解决突发流量及覆盖面的问题。

对于目前涌现出的很多大型互联网网站，往往采用自建 CDN 和租用 CDN 结合的方式。对于自建 CDN，可以降低成本，研发更符合自身业务特性的 CDN，提升用户体验，提高业务竞争力；同时，结合 CDN 租用，能够取长补短，提供自身不能研发的 CDN 特性，能够更好地响应业务突发需求，降低网络整体运营成本。CDN 自建和租用相结合的模式已经成为一种趋势。

自建与租用 CDN 相结合，还需要建立统一调度管理平台，屏蔽不同 CDN 的差异化，为融合业务提供基础。

以苹果公司为例，最初，苹果公司没有自主开发的内容传输平台和技术支持平台，并一直依靠 Akamai 和 Level 3 等 CDN 服务提供商开展苹果应用的传输服务，包括 iTunes 视频、iOS 和 OS X 的软件更新。苹果公司希望在所控制的平台上把用户体验做到极致，但在所有

用户体验中，对于内容向设备的传输方式，苹果没有进行控制。这种情况下，对于苹果的移动设备而言，由于系统的封闭性以及过度依赖第三方 CDN 企业，用户在使用应用商店下载应用和 iOS 等内容更新时，经常产生下载速度慢、体验不好的抱怨。

因此，苹果正致力于改变这种状况——自研 CDN，并通过将自己的 CDN 设备安放到 Comcast 和其他大型 ISP 的数据中心里，从而达到加快数据传输速度、提升用户体验的目标。

目前，苹果已经在美国和欧洲部署其自研 CDN，并对部分苹果应用提供了内容分发服务。但是，对于 iTunes、iTunes 电台和应用商店下载仍在通过租用第三方 CDN 向用户提供服务。

另一家大型互联网视频服务提供商 Netflix，最初的时候其流媒体视频服务和其他视频服务提供商一样，将 CDN 服务交付给第三方 CDN 服务提供商。但随着业务量越来越大，基于以下 3 个方面的考虑，Netflix 最终选择了建设与运营自有的 CDN，将应用以及用户的体验控制在自己的手里，为了给用户提供最佳的流媒体服务体验。

- 从业务发展的角度看，Netflix 的业务拓展速度，使得租用 CDN 服务提供商通过增加基础设施难以适应 Netflix 的业务范围。
- 从价格成本的角度看，基于增长的客户以及视频内容质量的提升，流媒体视频的流量越来越大，Netflix 租用 CDN 的服务成本也越来越高。
- 从业务模式的角度看，视频流正逐步取代其 DVD 租赁业务成为 Netflix 主要的收入来源与战略方向。

综上所述，无论企业采用何种方式租用或自建 CDN 服务，都需要根据企业自身情况、业务需求、客户分布、流量变化等多个因素加以考虑，在 CDN 租用和自建之间进行平衡。

（3）运营商纷纷切入 CDN 市场

由于越来越多的 ICP 通过 CDN 或者云计算平台接入网络，取代了原来所租用的 IDC 和互联网接入专线业务，为了占据未来竞争的高点，获取更多增值业务带来的利润，越来越多的运营商开始切入 CDN 市场。

例如，AT&T 改变其传统的技术策略，转而与 Cotendo 合作开展 CDN 服务。KDDI（日本的电信服务提供商），作为日本的第二大运营商，控股了韩国一家 CDN 公司 85%的股份。美国运营商 Verizon 则收购了美国本土一家著名 CDN 企业 EdgeCast。因此，运营商收购 CDN 专业服务提供商或与 CDN 专业服务提供商长期合作是一种趋势。随着电信网络运营商进入 CDN 市场的竞争，CDN 业务市场格局必将发生变化。电信网络运营商由于掌握网络资源，运营商切入 CDN 市场具有明显的带宽成本优势。不仅如此，运营商拥有深厚的用户群基础

以及多年的建设和运营网络服务的诸多优势，可以深入理解用户需求，为本网用户以及自有业务提供更优质的加速服务，并降低骨干网的传输压力。

因而，相较于互联网 CDN 服务商，运营商在业务方面有自身的优势，在竞争中会有不同的定位和发展空间。

6.2.3 CDN 网络发展趋势

在 CDN 业务和市场的发展过程中，技术驱动起到了很重要的作用。CDN 是整个业务的基础，是核心竞争力的来源，CDN 服务商需要考虑怎样才能够实现 CDN 的降本增效，怎样才能提供更优质的服务性能，怎样才能提供更丰富多样的增值服务。因此，CDN 服务商往往通过在传统 CDN 加速服务中融入创新技术，提高 CDN 的业务质量和市场竞争力。

（1）CDN 融合承载

前面已经提到，目前在线视频已经成为 CDN 最主要的业务，并且每年以 50%的复合增长率增长，此外，三网融合的到来也给 CDN、IPTV、移动互联网、PC 流媒体、互联网电视等新业务的不断发展带来了新的活力，使承载网络接受了大规模的网络数据流量。由于在建设过程中，只考虑单一业务场景，传统 CDN 节点无法实现多种业务融合的要求。为了实现多业务融合和多终端接入，必须对现有 CDN 进行升级，以满足未来多业务、多终端融合的市场需求。

如图 6-2 所示，根据国家统计局的数据显示，从移动互联网业务的发展速度上看，从 2013 年开始，移动互联网用户数超过了 PC 互联网用户数，说明用户对网络的依赖也逐渐转变到了移动互联网侧，因此，移动互联网业务会越来越普及，移动互联网的流量势必连年上升。

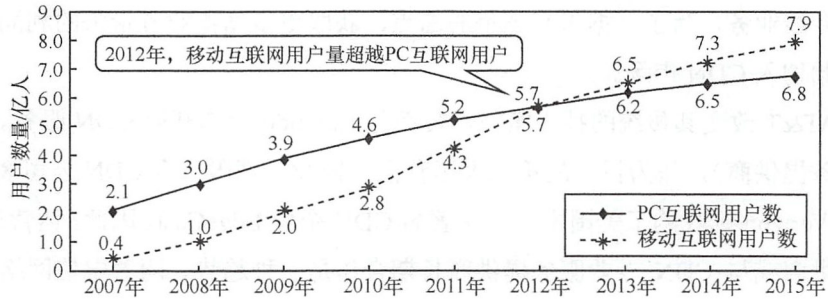


图 6-2　2007—2015 年 PC 互联网用户和移动互联网用户数量

移动互联网产业的快速增长，使得互联网应用整体规模得以跃升。如图 6-3 所示，购物、电商、搜索、视频、游戏以及音频等细分领域都获得较快增长，其中，移动支付成为推动移动端市场规模扩大的主要驱动因素。这些业务需求都将进一步转化为多业务、多终端的统一接入的需求，成为对融合承载 CDN 发展的驱动力。

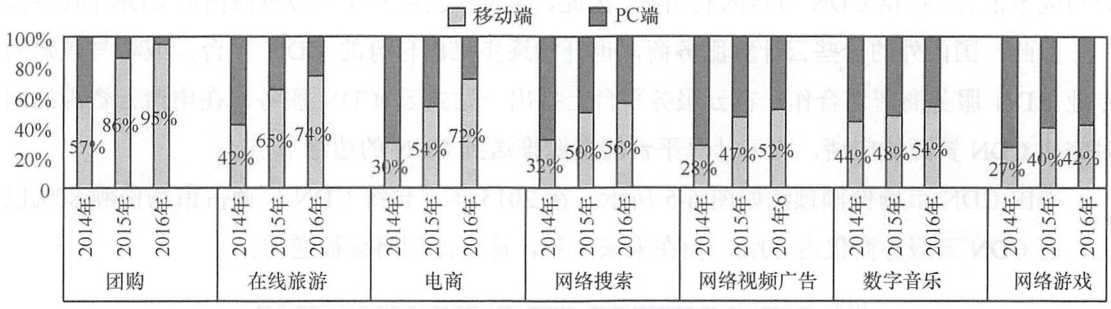


图 6-3　2014—2016 年互联网行业收入规模 PC 端与移动端占比情况

融合 CDN，作为一个承载多种业务的 CDN，能够实现承载与业务分离，支持自有业务和互联网业务的加速，在建设、运维、管理、经营等多方面，都有着巨大的优势。

图 6-4 介绍了从单业务 CDN 到融合 CDN 的演进方向。融合 CDN 将承载多种业务，实现承载与业务分离，网络对业务透明化，有效解决多网并存而不融合的尴尬局面，兼容流媒体视频服务、网站加速、网络游戏加速、文件下载加速和其他各类增值服务[7]。通过标准化的接口和业务通道，提升内容注入与存储以及内容分发等整体服务质量，并提供全面灵活的部署与互通能力。

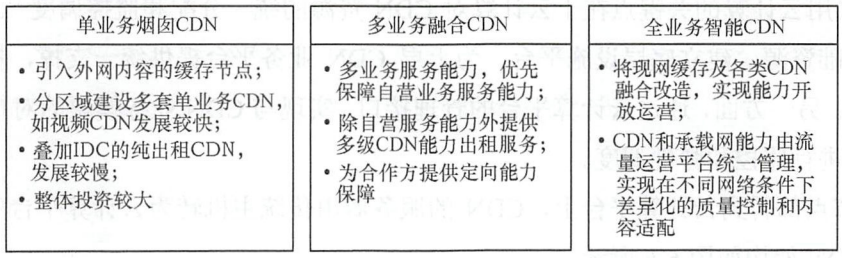


图 6-4　CDN 的演进方向

（2）CDN 与云结合

随着云计算的逐步普及，越来越多的 ICP 客户都成为云计算用户，在云平台上运行其网站或应用。云计算不仅能加速服务，还在存储与计算方面有着极大优势，CDN 与云计算结合，

会给 CDN 行业带来巨大的效益。

云服务平台能够提供基于网络的计算和存储能力，使得业务存储分离、节点资源池化，CDN 则提供分发能力，二者结合后形成能力的互补，大幅度提升 CDN 节点的带宽弹性。这样不仅能缓解用户请求高峰和突发流量的带宽压力，更能避免建设大量 CDN 节点导致的资源与成本浪费，提供 CDN 节点的利用率。因此，低成本也是云计算服务商拓展 CDN 的优势。

因此，国内外的一些云计算服务商，也开始逐步推出自身的 CDN 平台，或者与成熟的专业 CDN 服务商密切合作，在云服务平台上推出一站式云+CDN 服务，在申请云资源的同时完成 CDN 资源的申请，将云计算平台的资源推送到 CDN 的边缘节点。

中国 CDN 市场份额预测如图 6-5 所示，在 2015 年，传统 CDN 厂商占市场份额 80%以上，而 CDN 云服务商仅占 20%，但在未来 3 年，这个比例将会被逆转。

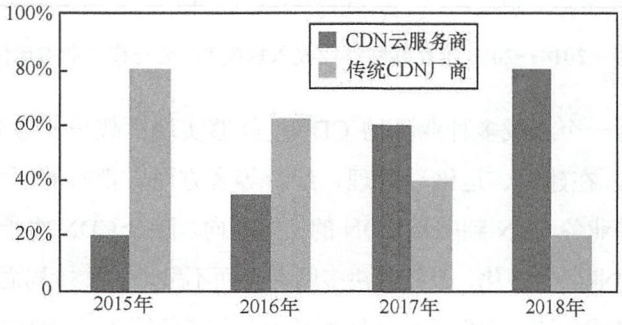


图 6-5　中国 CDN 市场份额预测

CDN 采用云计算的关键点在于云计算对 CDN 资源的统一分配和监控调度[1]。一方面，通过整合基础资源，建立底层设施平台，为上层 CDN 业务平台提供统一支撑，提升基础设施的利用率；另一方面，通过云计算平台的管理接口，实现与 CDN 业务模块的对接，对 CDN 的各类负载进行监控和弹性调度。

CDN 节点部署到云计算平台上，CDN 的服务器由传统主机转为云计算平台中的实例，即云化后 CDN 架构如图 6-6 所示。

CDN 云化后的架构可分为基础设施层、平台层（实例层）和业务层。基础设施层负责将固有的计算、存储、网络等资源虚拟化，形成共享的资源池。平台层（实例层）负责根据需要分配资源，提供给 CDN 使用，并开放接口完成云计算对 CDN 负载的监控和弹性调度。业务层主要部署 CDN 及其他各类服务[1]。

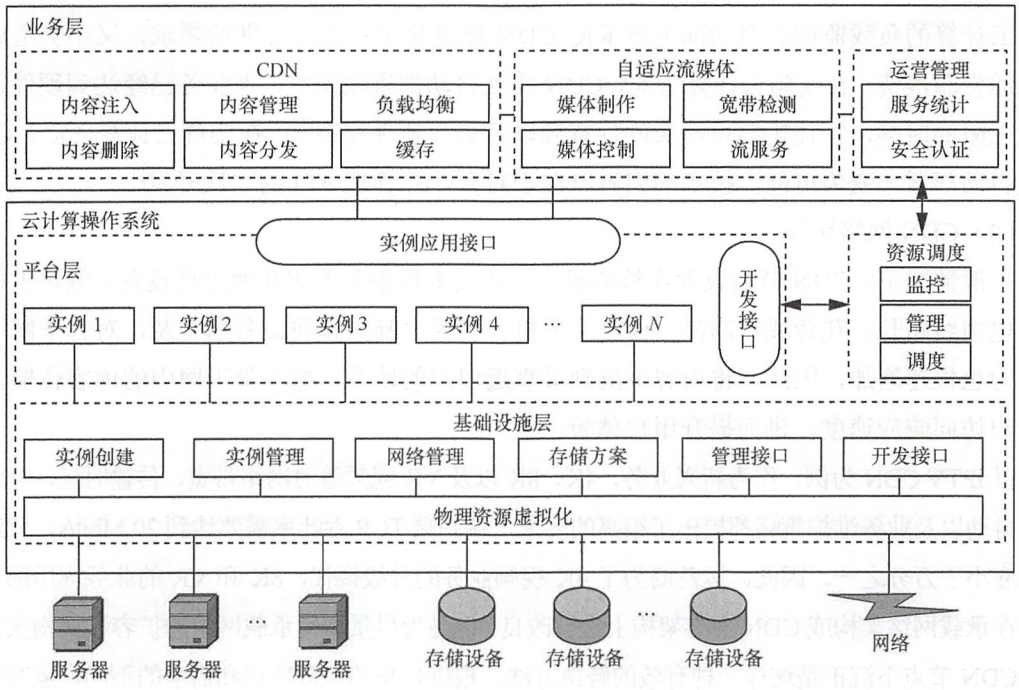


图 6-6　云 CDN 的架构

云计算与 CDN 的结合除了搭建云计算平台和部署 CDN 基本组件以外，还需要实现两个关键技术的突破，即实现云计算对 CDN 负载监控和自动部署（Auto Scaling）[1]，如图 6-7 所示。

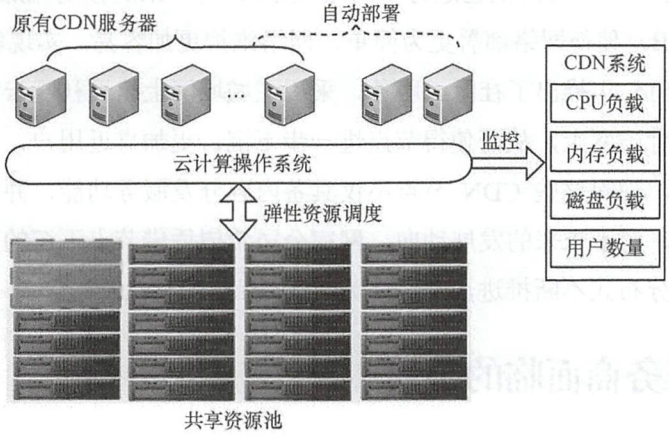


图 6-7　云 CDN 的负载监控和自动部署

云计算的负载监控和自动部署技术使 CDN 既能满足业务高峰期的需求，又不会造成低潮期的资源浪费。在实施云计算之前，CDN 实现自动部署很困难，当业务已经达到硬件设施性能上限的时候，往往需要再购买硬件资源以及进行软件安装[1]。在实施云计算之后，CDN 实现自动部署才变为可能，技术的突破口在于对云计算管理 API 的二次开发[1]。

（3）CDN 网络协同

一般情况下，CDN 节点设置在核心机房，涉及多种服务器和其他硬件设备，整体工程繁琐且电缆线路长，花费成本高昂。而随着手机、平板电脑等无线设备的普及，对骨干网的带宽压力也极速增加，因此，将内容下沉到最靠近用户的地方，减少骨干网中的内容传输，提高用户访问响应速度，进而提升用户体验。

以 IPTV CDN 为例，作为新兴业务，4K、8K 以及 VR 视频等对网络带宽、传输时延、分组丢失、抖动以及业务维护策略都提出了很高的要求，端到端 TCP 吞吐率需要达到 20 Mbit/s，分组丢失率应小于万分之一。因此，运营商为了 4K 视频业务的升级换代，8K 和 VR 的业务应用部署，需要在承载网络架构或 CDN 网络架构上进行改良。但是为尽量避免承载网络的扩容和架构大量变动，CDN 节点下沉正是这样一种有效的解决方法。根据自身的业务特点和需求的不同，运营商可以灵活调整网络和机房资源，确定是在城域网还是业务接入控制点等部署 CDN 节点下沉位置。

例如，中国电信在 CDN 网络规划中，对于用户分布密集的地区，在接入机房 BRAS 侧部署边缘 CDN，使得热点内容能够为本地区的用户提供就近服务，不仅能够保证 CDN 节点的服务质量，更能够很大程度地降低骨干网与核心 CDN 节点承载的流量负担。

中国移动在 CDN 网络协同上也进行了网络规划。针对未来的移动流媒体需求，移动 CDN 节点将下沉 eNode B，使得网络部署更为简单，网络维护更加容易，实现就近调度。

网宿科技在 2015 年推出了社区云服务，采用在城域网上建设社区云节点的方案，不将社区云节点部署在骨干网上，从而使得节点进一步下沉，更加靠近用户，就近部署。网宿科技的节点部署方案，使得这些 CDN 节点不仅具备内容分发服务功能，并且也具备存储、计算等“云”的功能。对于未来的发展动向，网宿公司希望凭借节点下沉的策略，促使内容分发方式从集中式向分布式不断推进，从而成为 CDN 架构的时代领跑者。

6.3 CDN 服务商面临的挑战

随着互联网用户的需求升级和网络规模的不断扩大，互联网承载的内容与数据流量已经

变得非常庞大。每天都会产生大量的数据流量进入网络，带来巨大的数据量冲击，随之而来的是对数据安全的保障需求。其中，流媒体业务的实时化对可靠性传输提出了更高的要求。同时，CDN 行业作为基础设施，需要通过不断创新和改善其服务能力，应对这些明显的变化，这在 CDN 业界已达成共识，使现有 CDN 服务商面临着巨大的挑战。

6.3.1　不可忽视的安全因素

最近几年，恶意网络攻击行为的规模逐步升级，互联网信息安全受到了很大程度的威胁，信息安全不容忽视，特别是需要满足电商支付、网银类用户极为苛刻的数据安全要求。此外，直播业务的数据传输，也需要 CDN 服务商对数据传输进行全方位保护，杜绝非法盗用问题的发生。此外，非法登录获取用户资料、篡改网页内容、插入恶意广告等现象也屡见不鲜，对 CDN 中的数据安全带来了巨大威胁。由于 CDN 节点的就近性部署原则，为保证不同地区的用户体验，往往在各个地区部署节点，因此，CDN 节点分布众多，使得删除异常信息的难度随之大幅度增加。这些网络攻击对加载时间以及对用户的最终体验的影响远超过其他因素，是应用性能的“无形杀手”。因此，加强 CDN 的数据信息安全是不容忽视的。

6.3.2　大数据流量的利用

全球每天产生大量互联网流量，经过 CDN 访问互联网的流量能够被采集、整理和利用。通过采集 CDN 中的数据流量加以分析统计，能够识别来自不同地域的用户使用的终端设备的类型和型号。此外，还能精确到基于特定 IP 地址的行为进行分析，能够准确判断特定 IP 地址是否存在对其他网站或者用户发起过网络攻击行为。另外，这些大数据流量对 CDN 中的业务发展也起到了重要作用，目前，利用 CDN 中的数据做到精准定位用户需求和偏好是整个行业研究的热点。利用大数据的流量精确地预测用户需求，获取用户访问的体验，完善整体业务的访问友好度，使 CDN 具备更精准的全局调度能力。总之，这些大数据信息被收集利用，目的是给智能逻辑判断打下一个技术储备基础。

6.3.3　推进统一的技术标准

CDN 行业规模不断扩大，使得不同服务商的 CDN 产品面临不同层面的互操作和对接需

求。长期以来，CDN 产品缺乏统一标准规范，开放性弱，各 CDN 服务商使用的私有接口、操作环境和系统状态等参数各不相同。这种情况下，CDN 管理需要采用基于多种类型的环境和产品不同的软件和工具来实现，就会导致需要投入大量资源和资金进行扩容或者改造。为了解决上述问题，CDN 与内容提供商之间的内容注入、不同 CDN 服务商之间的互联、CDN 与承载网之间的信息交换与对接等都需要不同层面的互操作标准化进行约束和规范，因此，CDN 标准化工作需要进一步推进。

6.3.4 定制化的技术创新要求

随着用户对 CDN 服务质量和可靠性保障需求的升级，市场上通用 CDN 产品的竞争进入白热化阶段，技术创新能力毫无疑问是打破这一僵化局面的关键因素。这时，只有研发差异、个性的定制化创新型 CDN 服务，才能满足互联网应用带来的新业务和新特性需求，并在 CDN 行业市场脱颖而出，避免其他同类 CDN 产品的挤压，甚至成为新型 CDN 产品的潮流推动者。例如，CDN 服务的“模块化”或者“部件化”的按需部署方案就是 CDN 技术创新的思路之一，有针对性地组合用户需要的模块，满足了用户的差异化个性需求。可以说，当前 CDN 市场的竞争，很大程度上来源于技术创新之间的竞争。CDN 服务商只有通过提升自身定制化的技术创新能力，才能够长期抢占 CDN 产业的高地。

第 7 章

租用 CDN 与自建 CDN 的选择

7.1 租用 CDN 与自建 CDN 对比

7.1.1 业务需求

随着智能终端的普及和 4G 网络的发展，互联网的业务需求日益增长，互联网应用从早期的网络新闻、搜索引擎等信息获取领域不断拓宽到视频、游戏等在线娱乐，微信、微博等即时通信以及团购、网上支付等电子商务的多元化领域。首先，这些互联网应用本身特点和用户对其核心需求不同，有着各自的加速需求。其次，通过不同业务逻辑的复杂性、需求变化频度、功能个性化要求以及响应时间的不同，评估通用的 CDN 服务商是满足业务要求还是需要自建 CDN。

通常，大型互联网公司对于核心业务采用自建 CDN 服务器，依托自建 CDN 针对性围绕核心业务进行优化，建立自身的核心竞争力，而不是采用通用化的 CDN 服务。例如，腾讯的微信、QQ 以及游戏等核心业务为自建 CDN，而门户网站、QQ 音乐等业务采购第三方 CDN；百度的搜索业务采用自建 CDN，而百度云以及部分视频业务采购第三方 CDN；阿里电商业务主要采用自建 CDN 的方案，但在“双十一”等电商促销活动时，采购临时 CDN 流量；视

频网站，大部分采用自建 CDN 方式，但对于世界杯、热播剧等流量峰值时刻会临时采购 CDN。因此，对于互联网业务量大且希望建立极致性能的核心业务，首先考虑自建 CDN。

7.1.2 CDN 与成本分析

（1）带宽成本

租用和自建 CDN 两者的带宽成本是不一样的。由于 CDN 服务商是向电信运营商大批量采购接入和托管服务，等到 CDN 技术增值之后，将 CDN 带宽出租给内容提供商，也就是说，使用 CDN 为内容提供商提供有质量保证和本地个性化的内容分发服务，同时，向内容提供商收取带宽和存储容量的费用，所以，在某些情况下，如果提供的业务总流量不大，但服务的用户分布很广，向 CDN 服务商租用第三方 CDN 比自己采购带宽更划算。相反，如果提供的业务总流量很大，用户分布相对集中，自建的成本就更划算。如果业务大部分很集中，但也有部分少量用户分布较分散，可使用租用与自建结合的方案。

（2）建设和维护成本

CDN 网络部署一次性成本较大，覆盖范围决定了 CDN 服务提供商的服务能力，在全国范围内部署节点服务器的成本投入对中小企业来说是个很难逾越的壁垒，高额的前期投资，让这些企业望而却步。同时，需要耗费一定的时间来实现自建 CDN，但是购买服务就可以做到“即买即用”。

由于大型网站或互联网企业在自行建设 CDN 系统中技术研发、更新周期、应用需求实现、运维管理等因素限制了企业大规模开展自行建设 CDN 系统，虽然 CDN 的工作原理不难，但要求提供商为用户提供售前技术咨询、网站加速、流量控制和实时统计分析等系列服务，也都需要专门的开销来支撑。

纵观自建 CDN 的历史，也就是一些资本雄厚的厂商能够支撑自建的开销，自建需要非常高昂的建造成本以及运营成本。

7.1.3 CDN 租用与自建结合

在互联网应用和移动互联网技术日新月异的今天，CDN 也要对网络应用的适应性做出快速响应。在技术研发能力和业务运营模式上，有着对 CDN 很高的要求。这就需要一个良好的互联网成本模型作为支持，使所有类型的用户都能得到更好的服务体验。中国大型互联网

公司在互联网成本模型的支持下，非常倾向于自建 CDN，而且包括阿里云在内，已经由自建自用发展到提供商用。

对于企业采用何种方式采购或自建 CDN 服务，往往需要结合企业自身情况、业务需求、客户分布、流量变化等多个因素加以考虑。企业自建 CDN 与采用外包 CDN 优劣比较见表 7-1，可以看出，自建 CDN 和租用 CDN 各有所长。对于很多互联网网站，发展初期业务量较小，往往采用租用第三方 CDN 的方式，但随着带宽流量大幅度提升，互联网带宽也达到一定规模，必然会产生自建 CDN 的需求。目前，美国更多是利用第三方 CDN，而国内，随着 CDN 市场规模的加大，商业 CDN 未来也会出现更多形态。在未来，自建 CDN 和第三方 CDN 的整体发展会齐头并进。

表 7-1　自建 CDN 与外包 CDN 优劣比较

	优点	缺点
自建 CDN	• 掌握 CDN 所有权、控制经营风险； • 灵活快速响应； • 针对业务做优化	• 建设 CDN 前期资金投入较高、建设周期长； • 覆盖网络范围较窄、难以实现全网加速； • 非主营业务、技术研发和储备不及专业服务商
外包 CDN	• 节约高额的前期投资； • 按需采购、避免资源冗余； • 腾出精力聚焦核心业务	• 完全依赖第三方服务商存在经营风险且缺少议价能力、不易控制成本

除了全部自建 CDN 或全部租用 CDN，还有第三种选择，即自建与租用 CDN 混合的方式。混合方式是采用自建和租用混合的方式对自己的 CDN 服务进行整体布局规划，包括地域、业务和峰值分担等混合策略。地域混合式是指内容供应商自主采购部分地域的网络服务，其余补充的地域采用租用的方式；业务混合式是指内容供应商自行开发核心业务的分发，而将其余的服务对外租用；峰值分担是指内容供应商日常稳定流量需求已满足，遇到特殊峰值或流量波动时，在自身带宽无法满足的时候临时租用。目前，CDN 自建和外包混合的模式已经成为一种趋势。

7.2　租用 CDN 的选择

对于互联网服务提供商而言，改善和确保服务质量通常是第一个需要考虑的因素。为保

证服务质量，在选择 CDN 服务商时，需要选择最能保证质量的 CDN 运营商。但由于互联网应用的复杂性，在选择 CDN 服务商时，往往很难选择一个全面满足自身要求的运营商。因此，互联网服务提供商应根据自身应用的特点，合理进行 CDN 的选择。

一般来说，选择 CDN 主要考虑以下要素。

（1）CDN 提供的服务类型、功能

不同的 CDN 运营商提供的 CDN 服务类型与功能是不一样的，如有些 CDN 只能提供静态内容的加速服务，而一些 CDN 则可以为动态内容提供加速服务。互联网服务提供商应根据自有业务的需求，首先明确哪些内容需要用 CDN 来加速，哪些不需要加速。

（2）CDN 容量与分布

目前没有任何一个 CDN 运营商可提供全球的由电信运营商网络全覆盖的 CDN，各 CDN 运营商只能根据现有客户的特点，选择在主要的运营商内进行 CDN 节点部署。因此，不同 CDN 在不同的宽带接入环境下，用户的服务质量是不一样的。互联网服务提供商应根据自有业务的用户群分布选择合适 CDN，某些情况下，可能还需要同时选择多个 CDN 共同承载与加速业务应用。

（3）CDN 运营维护能力

CDN 运营商提供给客户的运营分析、故障定位能力不一，如部分 CDN 运营商可为其客户提供多种图形化、自动故障定位的运营维护能力，部分可提供相关 API，可与自有运营维护系统集成，形成端到端的业务质量监控系统。互联网服务提供商应根据自身运营维护的要求，考虑 CDN 是否满足自要求。

（4）CDN 价格

CDN 价格是众多互联网服务提供商选择 CDN 的重要考虑因素，CDN 有多种计费方式，在不同的电信运营商网络内，价格也不一，互联网服务提供商可根据自有业务的流量特点，选择合适的计费模式下的不同 CDN。

7.2.1 CDN 服务类型与功能

现有 CDN 根据其承载的内容与应用可区分为以下类型 CDN：静态小文件服务、静态大文件服务、流媒体服务服务、动态网站加速服务以及全站加速服务。不同 CDN 服务商提供的服务能力是不一样的，比如，传统的 CDN 服务商服务类型会比较全面，新兴的 CDN 服务

提供商可能在某些领域，比如流媒体直播等方面做得比较好，租用 CDN 时需要根据自身业务需求和目标来进行选择。

对于选择 CDN 而言，可为每种业务填写表 7-2 的表格，分析自身的业务需求。

表 7-2　CDN 业务需求

	静态小文件	静态大文件	流媒体服务	动态网站加速服务	全站加速服务
网络覆盖					
网络带宽					
资源能力					
自助服务					
运营能力					
支撑能力					
增值能力					

7.2.2　CDN 容量与分布

CDN 厂商之间的竞争很大程度上就是节点容量与分布的竞争。CDN 只有为用户提供就近服务才能提供较好的服务质量，所以 CDN 覆盖面越广，各节点容量越大，相应的服务质量才越高。但是，各个 CDN 服务提供商在各个运营商内获取资源的能力是不一样的，所提供覆盖的质量也是不一样的，有的服务商在北美、欧洲覆盖比较好，有的服务商在亚洲覆盖比较好。而且服务商往往所提供的各地区的 CDN 服务能力价格也是不一样的，如 Level 3 就是参照不同区域提供不同的服务质量为 CDN 定价的。因此，租用 CDN 资源时需要分析业务覆盖的核心区域来选择性价比最优的方案。

7.2.3　CDN 运营维护

租用 CDN 需要为客户提供维护手段与方法，以便用户能够自行配置与调整。一个完善的 CDN 维护系统，可以帮助客户快速定位服务质量问题的根源，提高服务质量，提升用户满意度。因此，方便运营与维护也是租用 CDN 需要考虑的重要因素。

一般而言，CDN 运营商会向用户提供多种自维护工具与能力：门户、用户管理、资源管理、监控系统、线路调度配置、内容管理、配置管理、数据分析、计费统计与管理、自动部

署等功能，这些功能可与自有运营维护系统集成。

- 门户：向互联网服务提供商系统管理员提供用户访问系统相关管理的人机操作界面和访问入口[8]。
- 用户管理：用于完成用户的新增、删除，用户信息修改、密码修改、密码重置、用户状态设置、用户信息查询等操作[8]。
- 资源管理：用于对整个内容分发网络的服务器、交换机、存储、节点带宽等资源进行管理、规划等，资源管理系统需要将各个节点的资源进行应用划分，不同节点的相同应用类型的资源可以为同一种业务所使用。
- 监控系统：对各类资源（设备、带宽）、应用程序进行监控，提供对各类资源及应用程序的基本信息监控、故障监控、性能监控，系统提供最小粒度的关键性能监控，从而了解关键性能的变化情况，监控系统的功能包括资源基本信息采集、资源告警信息采集、监控信息预处理、故障检出规则（基于告警信息分析得出故障结论的规则）管理、故障分析、故障定位、告警展现以及告警处理[8]。
- 线路调度配置：负责将用户对网站的访问引导到离其最近的 CDN 节点上，线路调度配置除了根据 IP 地址库进行调度外，还可配置故障调度、节点流量跑满调度、节点服务质量调度以及攻击调度。
- 配置管理：可为用户提供自行修订 CDN 服务配置的能力，如缓存配置、访问控制、日志格式等。
- 内容管理：对用户网站缓存在 CDN 节点上的副本内容进行管理，支持对文件和目录的清除，支持文件的预取加热。
- 数据分析：负责将用户在 CDN 节点的访问进行记录并分析，通过分析，可以获取有价值的信息以供用户运营参考，可以获取的信息如运营商流量比例、省份流量比例、PV、热门文件、访问来源以及服务器日志等。

7.2.4 CDN 价格

CDN 价格是每个 CDN 服务商关注的方面，但并不是最低价格的 CDN 就是最好的 CDN，因为不同的 CDN 功能、性能、管理、服务等能力是不一样的。而且由于 CDN 存在不同的计费方案，所以互联网服务提供商应根据自有业务的流量特点，综合 CDN 的功能性能等方面

合理选择。

现有 CDN 一般提供两种计费方式：带宽计费和流量计费。带宽计费是按用户业务每天的带宽使用进行计费，流量计费则是按用户业务每天使用的流量进行计费。带宽计费一般有 95 带宽计费、月均带宽计费等方式。

95 带宽计费是将每 5 min 作为一个带宽统计点，并将一天共 288 个统计点按照从大到小排序，去掉最高 5%的统计点，剩余最大值即当天的 95 带宽；日月均带宽是取 CDN 每日 288 个带宽统计点，对每一个有效天的峰值取平均值，得到计费带宽。目前 95 带宽计费方式比较常见。

流量计费比较简单，即统计用户每月使用的总流量，根据流量进行计费。

CDN 服务提供商可以根据自身业务形态选择合适的计费模式，也可以通过计算带宽利用率来选择适用的计费模式。带宽利用率即使用的带宽除以峰值带宽获得的值，如果带宽利用率较高，业务的曲线较为平稳，建议采用带宽计费方式；带宽利用率较低，表明业务的曲线波动较大，建议采用流量计费方式。

CDN 计费系统负责合并统计各个加速域名的计量信息，并按照一定的规则生成最终的计费信息，计费系统通过部署在各个服务器上的代理程序定期进行计量信息采集并汇总。用户付费方式根据具体服务情况而定，可以一次一付，也可以按月、按年付费。

7.2.5　多 CDN 租用

前面已经提到过，对于较大型的互联网企业而言，随着互联网和大数据的发展，各厂商为了实现用户跨地域访问网络获取资源的快速、稳定、高效，分别自建 CDN 网络分发系统，希望全面解决由于网络带宽小、用户访问量大、网站分布不均而产生的用户访问资源响应速度问题[9]。自建 CDN 是节省总体投资、提高业务质量的较好方案，但这些企业的自建 CDN 基于成本的考虑，并不会在全球部署，也难以支持在业务突发期的流量。为此，这些企业往往需要租用第三方 CDN 作为自建 CDN 的补充。

对于全部租用 CDN 的互联网企业而言，由于受到各国政策或者网络条件限制，每家 CDN 厂商都有各自优点和一定的不足之处[9]。根据客户定制需求，依赖单一 CDN 即满足所有需求也是非常困难的。如果单独使用一家 CDN 系统服务用户，未能达到调度最佳服务效果，就需要进行流量在多 CDN 间的调度与管理[9]。因此，多 CDN 租用或者自建 CDN 和租用 CDN 结合使用

时，都需要建设流量调度系统（TMS），实现用户访问在不同 CDN 之间的调度和流量分配。

7.3 自建 CDN 的选择

自建 CDN 可以有两种建设方案：一种是自主研发、部署和运营，另外一种是采购成熟 CDN 设备商解决方案进行建设部署，自行运营维护。

7.3.1 开源 CDN 软件与商业 CDN 软件对比

当互联网企业开始考虑自建 CDN，不管是一部分的流量由自建 CDN 来承载，还是全部流量都交由自建 CDN 来承载，都需要首先考虑自建 CDN 的建设模式。

自建 CDN 可以有两种建设方案：一种是采用 CDN 设备提供商提供的商业 CDN 解决方案，另一种是基于互联网开源软件自行研发部署 CDN 解决方案。

以华为公司的商业 CDN 产品为例，其提供 CDN 内容库、缓存、边缘流媒体、GSLB 等商业 CDN 设备。这些 CDN 设备软硬件是一起提供给客户的，其可靠性高、单机性能也非常高。但是，这类厂商 CDN 能力研发周期相对较长，缺乏开放性，用户无法自主升级更新。

而基于开源软件搭建的 CDN 解决方案也有不少案例，如 Netflix 就在全球利用开源软件搭建了其自有 CDN，承载了全球的视频用户。开源软件自身往往提供了灵活配置和修改的功能，在开源特性不能满足业务需求时又能够根据业务特性研发自身的插件或模块一起协同工作，能够更好地满足定制化需求。

将开源和自主研发相结合，会对 CDN 有更好的可控性，一些系统优化问题可以从底层解决，系统扩展性也更高。但基于开源软件搭建 CDN 系统，往往需要投入大量人员进行软件的优化、硬件适配、系统维护等工作，所以从总成本来看，并不是自主研发方案就一定会比商用 CDN 解决方案更低，所以互联网应用提供商还需结合自身情况合理选择相应的方案。

7.3.2 业务需求变化

每家厂商自建 CDN 的需求基本都是基于以下几种考虑。

- 成本：在体量不大的时候，自建 CDN 就是投入远大于租用，当体量增长之后，可以

考虑通过自建 CDN 降低成本。

- 定制化需求：第三方公司的定制化需求无法完全满足于自身业务发展的需求，且效率远远不够，自建可以根据自己的业务需求做更多的定制化业务。
- 备份和可控：自建 CDN 风险可控，在 CDN 出现问题的时候也可以作为备份，对于业务风险的管控有较大的作用。
- 扩展对外服务：CDN 作为基本的流量入口，大型企业通过 CDN 将数据库等资源紧紧地抓在手中，更精准地满足用户需求。

但是，自建 CDN 还需要根据自身业务需求变化是否频繁、业务需求是否特殊的个性化需求、业务需求量是否足够大来考虑是直接采用 CDN 设备商方案建设，还是自行研发建设。

例如，CDN 里面业务量比较大的视频业务，其实现协议标准，业务需求较为稳定，市场上能够提供成熟视频 CDN 产品的 CDN 设备厂商较多，如国内的华为、中兴通讯、烽火，国外的爱立信、思科等，因此，许多企业都采用 CDN 设备商的解决方案进行自建，如中国电信的 IPTV CDN、中国移动的内容分发网络等。

而互联网企业往往其业务需求变化快速，需要极强的快速响应能力，能够随着用户资源访问的变化特征快速灵敏地协调资源以及快速根据业务逻辑变化优化 CDN 服务逻辑。CDN 设备商提供的解决方案往往因为开发流程较慢且会更多地考虑需求的通用性，而无法满足互联网公司的需求。因此，更多大型互联网公司会采用自主研发的方式自建 CDN。

7.3.3　开发和维护能力

基于开源软件自建 CDN 方案的针对性强，可以与自身业务结合得更加紧密，有针对性地进行优化，也可以更好地掌握控制权及技术，以便灵活地随时进行开发调整。

然而，自建 CDN 需要企业具有较强的自主研发能力和维护能力，虽然说是基于开源软件，可是开源软件的安全性、稳定性往往是需要通过反复的测试和开发去完善的。而且因为 CDN 系统涉及了内容库、RR 调度、缓存、网管等多个部件，每个部件都基于不同的开源系统实现，需要投入大量的人力去开发和维护网络，才能使其达到商用的可靠性和稳定性标准。

如果没有一支专业化的技术团队，是完全不能够胜任的，只有通过加强技术研发力量，及时把握业务需求，才能在自建 CDN 过程中取得成功。

第 8 章

租用 CDN 实施的考虑与评估

租用 CDN 服务时需要考虑 CDN 本身的服务能力，还应该结合用户使用需求，不同类型和行业的网站对 CDN 服务的需求也是不一样的。下面结合互联网服务提供租用 CDN 实施步骤，详细介绍租用 CDN 的用户需求的确认、验证以及实施过程。互联网服务提供商租用 CDN 实施步骤如图 8-1 所示。

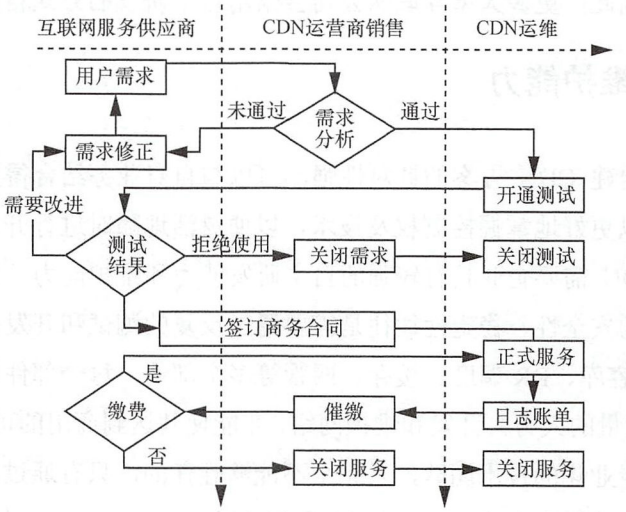


图 8-1　互联网服务提供商租用 CDN 实施步骤

由图 8-1 可见，租用 CDN 主要实施步骤包括以下 3 个部分：需求确认阶段、CDN 测试阶段、正式服务阶段。其中，测试阶段是用户需求的验证及多家服务商服务对比的过程，正式服务阶段是用户需求的实施及持续监控过程。

8.1　CDN 需求确认

在需求确认阶段，互联网企业作为 CDN 的用户，需要与 CDN 服务提供商销售人员沟通清楚需求，并由 CDN 服务商销售人员确认是否能满足。为便于确认需求，双方应以需求信息表作为确认的依据，CDN 服务提供商指导用户填写。需求信息表应包括用户对 CDN 的需求，包括加速域名、业务类型、协议、服务区域及防盗链等信息。典型 CDN 需求信息见表 8-1。

表 8-1　典型 CDN 需求信息

用户信息			
客户名称	*		
用户名	□已有用户　　用户名为： □新用户　　用户名为：		
销售负责人	*	客服项目负责人	*
产品选择			
基础产品	□网页加速　□图片加速　□HTTP 视频加速 □动态加速		
增值产品	□防盗链　□智能负载均衡　□IDC 主机托管		
服务需求			
频道数量	*	频道属性	□测试　　□商务
频道名称（域名）	*（如为多个频道同一 IP 地址请在此行填写所有频道名称）		
频道源站点对应 IP 地址	□中国联通 IP 地址：　　□中国电信 IP 地址： □海外 IP 地址： □其他（请说明所属 ISP，如果是双通或者 BGP IP，也请填写在这里，如果是多个 IP 地址，请注明主备关系或者轮询关系）		
回源站点的方式	□域名方式　　域名为： □固定 IP 地址　　IP 地址为：		

（续表）

业务流量分布	* 中国联通和中国电信业务带宽比例： * 其他运营商分布要求： * 重点关注地区： * 海外流量分布要求： * 预计上线带宽情况： * 用户期望节点分布：
客户源站点情况	□有防火墙限制 □有连接数/带宽限制　　具体连接/带宽限制数量： □有特殊端口　　具体端口号（如有多个，请用逗号分隔）： * 客户为其源站在 IDC 采购的带宽为（单位：Mbit/s）：其他： 客户办公室接入情况： * 所属 ISP：　　* LDNS IP 地址： * 带宽：　　* 接入类型：
刷新策略	* 选用的刷新策略： □遵循源站点 Header 设置 □在缓存设备上设定过期时间 □采用刷新接口 □手动在门户的刷新页面上进行主动刷新 * 过期时间规则详细描述：（如果选择了在缓存设备上设定过期时间的选项，必须填写本项） 文件类型或 URL 描述： 不缓存文件类型或 URL 描述： 过期时间： Default：处理策略（过期时间或者不缓存）：
压缩	□支持压缩 压缩方式：□判断 HTTP1.1 和 1.0　　□不判断 HTTP1.1 和 1.0 □含有 vary 参数 □含有 transfer-encoding:chuncked 的 header □其他：
日志	□需要日志 □对图片进行过滤 *　日志格式：□W3C　　□Squid　　□NCSA *　下载方式：□登录门户　　□日志下载 API □其他要求：
文件情况	* 文件大小最大值： * 文件大小平均值（多数文件的平均大小）： * 文件个数总数： * 大小文件所占比例（大小文件以 1 MB 为区分点）： * 文件存储总量： □其他要求：

（续表）

和源站点的配合	□是否需要在源站点内容无法获取时，缓存直接对外提供服务 *　本功能开启时间： *　本功能终止时间：
计费	□与原计费单元绑定 □新计费单元　新计费单元名称为：
频道监控链接地址	□采用标准监控对象 □采用非标准监控对象　非标准监控链接为：
防盗链策略	□refer 防盗链 □cookie 防盗链 □验证算法防盗链 □其他防盗链策略：
其他需求说明	

8.2　CDN 测试验证

当租用双方明确了 CDN 承载加速需求后，CDN 运营商应向其 CDN 运维人员提供需求表，由运维人员进行 CDN 接入测试配置，主要配置的内容包括：加速的域名、服务类型、回源方式、加速区域、缓存规则、查询串缓存设置以及访问控制。配置完后，用户服务就切换到 CDN 进行测试验证，并根据测试结果考虑 CDN 服务提供商的选择。

8.2.1　CDN 服务切换

当用户通过浏览器输入域名，向网站发起访问请求时，DNS 服务器会对该域名进行解析。

在对网站访问未加入 CDN 服务的传统接入模式下，DNS 服务器会返回给浏览器该域名对应的 IP 地址，浏览器使用该 IP 地址向网站服务器发起内容请求，网站服务器将用户请求的内容返回给浏览器。

但是，引入 CDN 后，会通过 DNS 配置切换到 CDN 服务，最终域名的解析权会由 DNS 服务器交给 CNAME 指向的 CDN 专用 DNS 服务器。此时，CDN 的 DNS 服务器将 CDN 的 GSLB 的 IP 地址返回给用户，用户再向 CDN 的 GSLB 发起内容访问请求，CDN 的 GSLB 会选择合适的节点为用户提供服务。

8.2.2 CDN 服务质量测试

在租用 CDN 时，用户面临着如何选择 CDN 服务提供商，以获得良好的 QoE 的问题。为此，互联网服务提供商企业在对 CDN 进行功能、容量、分布等因素分析后，可对意向 CDN 服务商提出测试要求，通过把服务切换到多家 CDN 供应商所提供的服务上进行测试评估，再根据测试结果选择相应的 CDN 服务商。

一般而言，CDN 服务提供商均会向用户提供测试服务，互联网服务提供商的用户只需要把某些测试页面或应用接入服务切换到 CDN 中，就可以通过 CDN 提供服务，从而可进一步测试 CDN 承载的性能。在选择 CDN 服务商时，鉴于 CDN 服务是一个黑盒服务，有其特殊性，同样往往需要了解到下面的几种 CDN 性能的情况：

- CDN 部署后的展示性能为多少，是否有提升，提升幅度多大？
- 提供了多少主机节点？
- 这些主机节点分布在哪些区域和运营商？
- 每台主机节点的性能如何，可用性是否稳定？
- 目标客户是否正确命中对应主机节点，或匹配度是否合理？
- 单台主机节点的覆盖范围或承载比例如何？
- CDN 节点与源站的同步效率做得如何？
- CDN 对元素的发布技术是否提供到位并长期有效？
- CDN 节点故障源站没有日志，如何对其进行及时报警？

这些性能的评估，通常建议采用国内外比较知名的第三方评测公司，如国外的 Gomez，国内的基调、博瑞等公司的产品，由它们来模拟全国乃至全球的测试机发起监测，对比加速效果，根据加速对比报告来选择合适的 CDN。

第三方测试产品基本原理为模拟用户的访问行为，进行诸如访问网页、文件下载等测试，记录测试中的各项数据（如 DNS 时间、连接时间、首页打开时间、下载时间等），通过这些指标对 CDN 服务质量进行监控，准确地反映服务质量的实际情况。

为满足 CDN 测试的准确性要求，往往要求测试产品可在全球各大电信运营商均有相关数量的测试客户端，并可进行长时间稳定性测试。在建立测试任务时，应根据自身业务的特点，选择合适的测试服务，填写测试任务。测试时，测试点数越多，时间越长，测试的精准

性就越高，可在合理范围内选择测试任务内容。

目前第三方测试业内公认的各性能指标如下。

（1）响应时间

对于 CDN 测试，响应时间及可用率应重点关注，其中响应时间又可分为平均响应时间、中位响应时间、75%用户最差响应时间、90%用户最差响应时间等，某次测试的指标见表 8-2。

表 8-2　CDN 关键指标

类别	取值
平均响应时间/s	0.654
中位响应时间/s	0.386
75%的响应时间/s	0.640
90%的响应时间/s	1.200
可用率	99.989 483 65%

对于响应时间，可细分为 DNS 时间、建立连接时间、SSL 握手时间、重定向时间、发出请求时间、收到第一个分组时间、客户端时间、内容下载时间等。

DNS 响应时间组成见表 8-3。DNS 响应时间主要考量根据用户来源判断分配至服务器的智能 DNS 解析性能，建立连接时间主要考量服务器硬件的响应性能，收到第一个分组时间主要考量动态或回源的性能。而对于页面加载，还应根据不同需求选择测试页面开始渲染时间、页面首屏时间、页面完全加载时间以及文件内容下载时间等。互联网服务提供商可根据上述这些细分的响应时间分析业务可改进的地方。

表 8-3　响应时间组成

技术指标	好	正常	差	备注
DNS	<0.18 s	0.18~0.3 s	>0.3 s	DNS 解析时间，用户访问页面的第一步
建立连接时间	<0.15 s	0.15~0.3 s	>0.3 s	建立连接时间是指 IE 浏览器和 Web 服务器建立 TCP/IP 连接的消耗时间
重定向时间	无	<0.1 s	>0.1 s	重定向时间是从收到 Web 服务器重定向指令到请求 Web 服务器的第一个元素之前的消耗时间
收到第一个分组时间	<0.2 s	0.2~0.4 s	>0.4 s	第一个分组时间是指 IE 浏览器发送 HTTP 请求结束开始，到收到 Web 服务器返回的第一个数据分组的消耗时间

（2）节点性能

切换 CDN 服务后，CDN 提供的节点数量、节点的可用性及节点的承载量都可以在 CDN 系统中体现。最理想情况下 CDN 各个节点性能都是一样的，但实际部署应用中很难达到这种理想状态，图 8-2 所示为 CDN 节点性能测试结果。

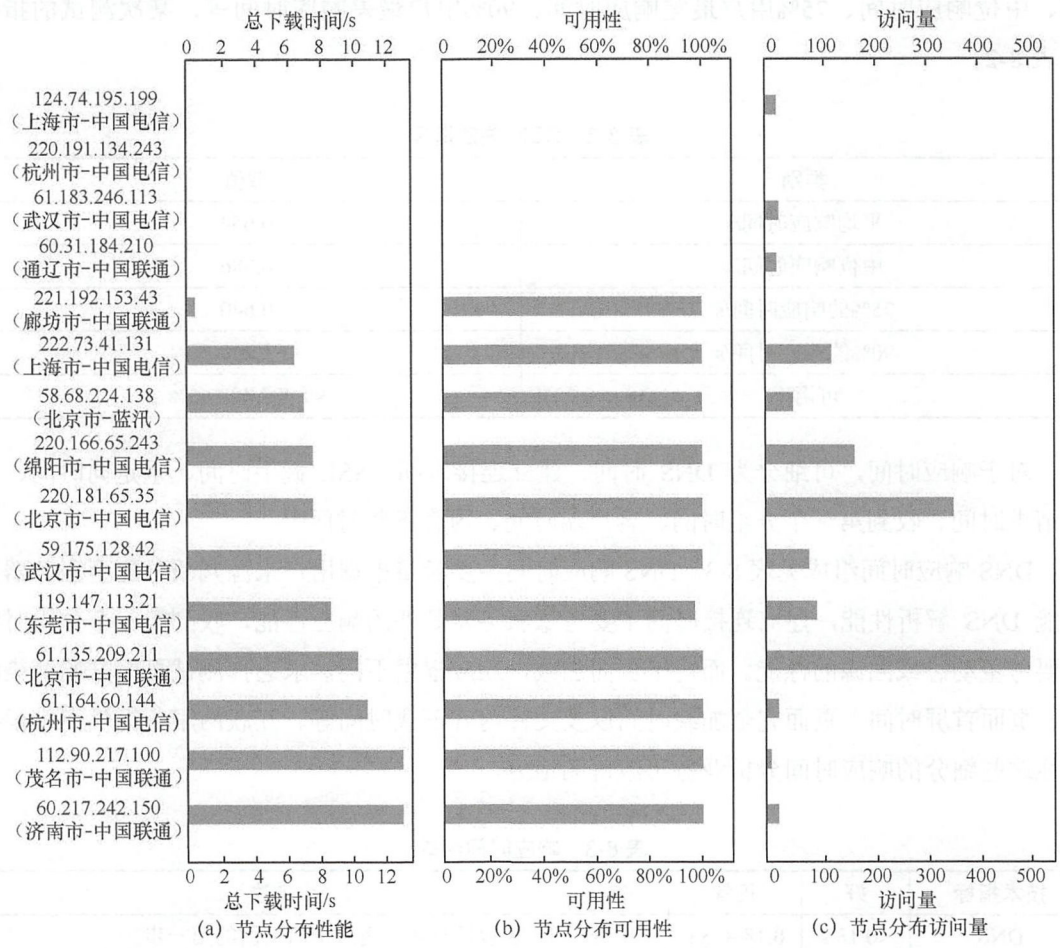


图 8-2　CDN 节点性能测试结果

（3）服务错误率

服务错误分为元素级和页面级两种，其中，页面级错误造成的影响高于元素级错误。服务错误率是指总的测试期间，错误次数的占比。表 8-4 说明了元素级错误一般只是影响用户

体验的友好度而页面级错误会直接导致无法服务。图 8-3 分别展示了不同 CDN 服务商节点的元素级和页面级错误测试结果。

表 8-4　服务错误指标

技术指标	好	正常	差	备注
页面级错误	<1%	1%~2%	>3%	页面出错直接导致用户将无法浏览
元素级错误	<2%	2%~4%	>5%	元素出错对用户友好度和体验影响较大

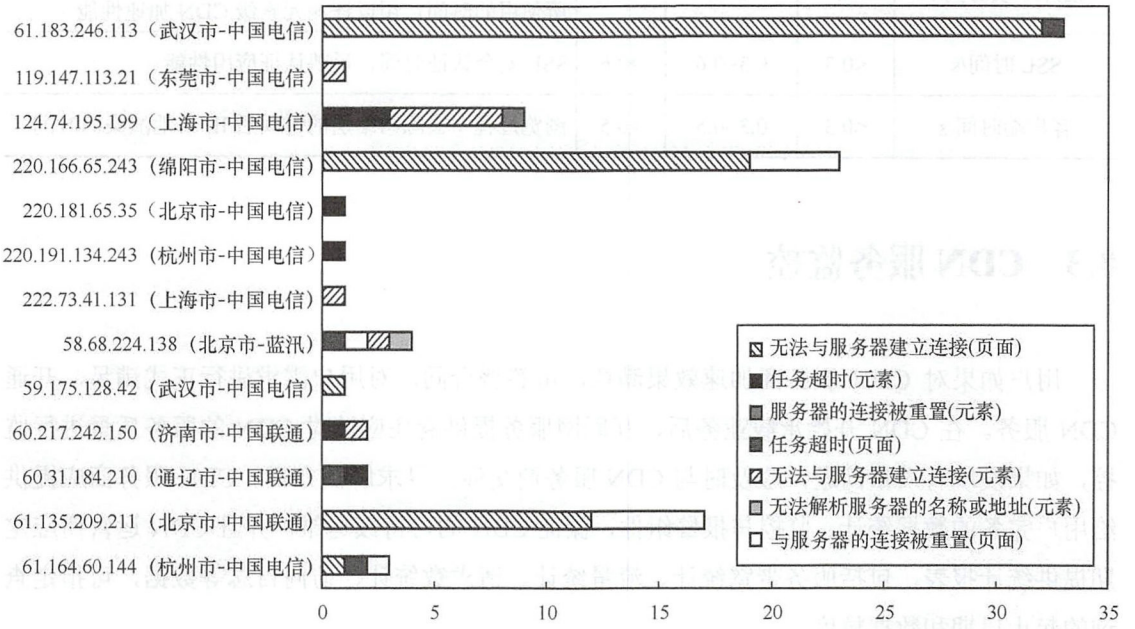


图 8-3　服务错误率测试结果

（4）应用层性能

网站应用层性能一般是指对网页下载、图片下载、SSL 应用加速等应用测试的完整响应时间。应用层性能对用户体验会造成直接影响，也是网站整体性能的表现，基于应用层测试的性能指标及优劣判断标准见表 8-5。

在测试正式开通后，用户可要求 CDN 服务商提供第三方的测试报告，证明其 CDN 加速的性能效果。测试报告可包括加速效果报告、流量报告等分析结果，并通过图表形式，对用户需求的验证结果进行汇总。

表 8-5 应用层性能

技术指标	好	正常	差	备注
可用性	>95 %	90%~95%	<90%	反映页面打开成功率，可用性过低影响站点形象
页面总下载时间/s	<10	10~20	>20	页面总下载时间是指页面所有内容下载时间，总下载时间主要表示的是页面的总体耗时，不同类型站点标准不同。现有标准仅仅作为默认参考
图片下载时间/s	<1	1~2	>2	图片总下载时间是指默认对 150 KB 大小的图片下载所使用的时间，用以评测元素级 CDN 加速性能
SSL 时间/s	<0.3	0.3~0.6	>0.6	SSL 安全认证时间，反映认证应用性能
客户端时间/s	<0.3	0.3~0.5	>0.5	浏览过程中去除网络层的时间后 IE 本地渲染的时间

8.3 CDN 服务监控

用户如果对 CDN 服务商加速效果满意，可签署合同，对用户需求进行正式满足，开通 CDN 服务。在 CDN 开始承载业务后，互联网服务提供商还应针对 CDN 的服务质量进行监控，如果发现有质量问题，应及时与 CDN 服务商交涉，寻求解决方案。CDN 服务商应提供给用户完备的数据统计、监控与报警组件，保证 CDN 的可持续运维。并且 CDN 运营商应定期提供统计报表，包括服务带宽统计、流量统计、请求数统计、访问日志等数据，可指定查询的起止日期和数据粒度。

其中，CDN 运营商提供的基本粒度的关键性能监控，主要分为设备监控和服务监控。

（1）设备监控

设备监控主要包括对服务器的流量、负载、CPU 使用率、内存使用、磁盘空间使用等指标的监控（对于租用专用 CDN 适用）以及对交换机带宽占用情况等指标的监控。

• CPU 使用监控

当 CDN 服务器的 CPU 使用率过高时，说明服务器上的应用正在进行进程调度或者系统调用，因此需要耗费大量 CPU 开销，例如，大量的数据查询工作和排序计算就会造成 CPU 使用率过高。一个典型的 CPU 监控结果如图 8-4 所示。

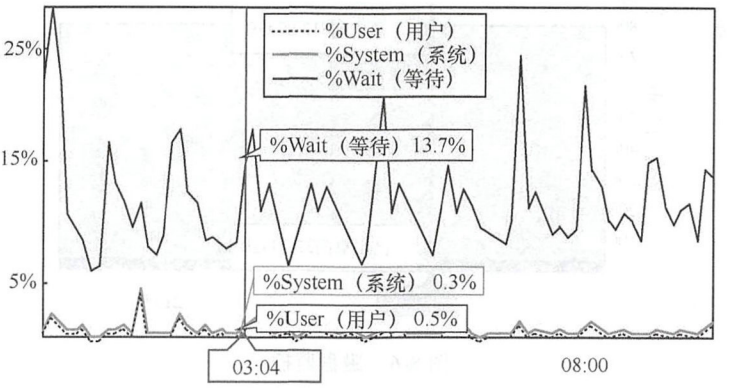


图 8-4　CPU 监控

• 内存使用监控

随着内存使用的增加，CDN 服务器的内存使用率就变得越来越高。其中包括一些虚拟内存的异常消耗，使得 CDN 服务器的响应速度严重变慢。因此，为了优化调整内存的使用策略，对内存的监控也是十分有意义的。一个典型的内存监控结果如图 8-5 所示。

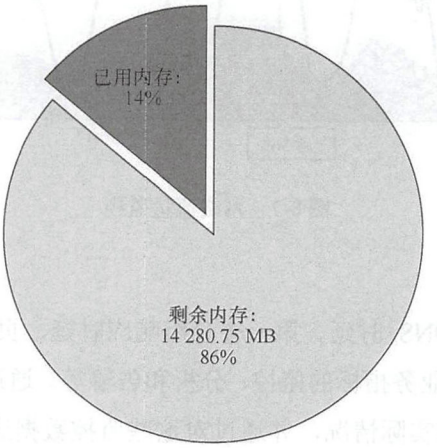


图 8-5　服务器内存监控

• 磁盘空间使用监控

监控磁盘空间的作用在于发现到底是哪些用户、文件或者应用程序占用了大量的磁盘空间，达到合理分配和优化磁盘空间的目的。一个典型的磁盘监控结果如图 8-6 所示。

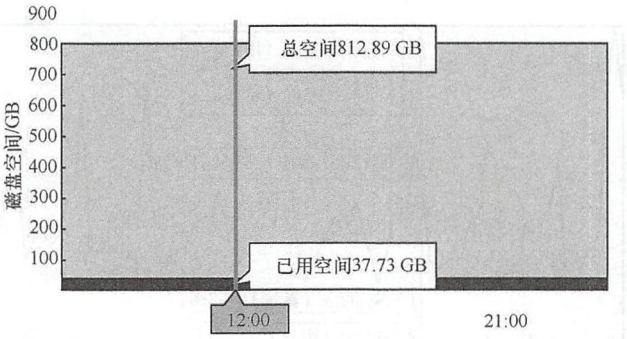


图 8-6　磁盘监控

• 网络流量监控

网络流量监控是判断 CDN 运行情况的关键因素，能够反映网络的运行状态，找出整个网络运行的资源瓶颈。当 CDN 承载的流量超过自身实际能力时，就会导致网络性能下降，一个典型的网络流量监控结果如图 8-7 所示。

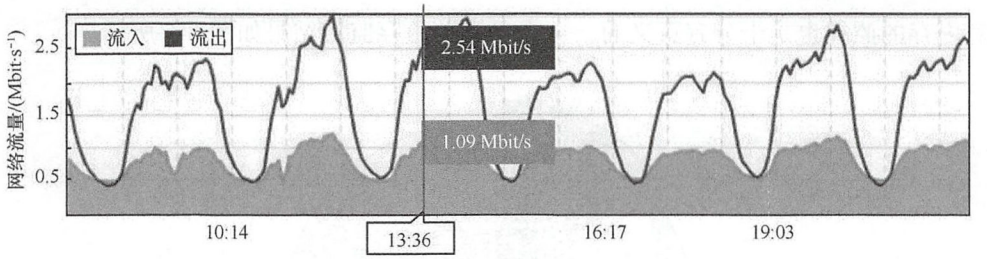


图 8-7　网络流量监控

（2）服务监控

服务监控主要包括对 DNS 时延、第一个分组响应时延、页面下载总时延、图片下载时延、服务可用率、卡顿率等业务指标的监控、分析和告警等。通过对服务质量进行实时监控，及时准确地反映服务质量的实际情况，并通过对这些监控数据进行优劣解读，分析出 CDN 目前的运行状态。具体监测指标与第 8.2.2 节类似，可进行参考。

8.4　多 CDN 租用调度

对于较大型的互联网企业而言，自建 CDN 是节省总体投资、提高业务质量的较好方案，

所以国内外众多互联网企业都在自建 CDN，但这些企业的自建 CDN 基于成本的考虑，并不会在全球部署，也难以支持在业务突发期的流量。为此，这些企业往往需要租用第三方 CDN 作为自建 CDN 的补充。

同时，各 CDN 服务提供商由于受到各国政策或者网络条件限制，每家 CDN 服务商都有各自的优点和缺点，甚至不同地区采用不同服务价格。如果全球单独使用一家 CDN 服务商的 CDN 产品，往往不能达到在全球各地提供最佳服务、实现最优性价比的效果[9]。因此，有的企业从服务质量、可靠性和价格的角度出发，也会选择同时采用多家 CDN 服务提供商的服务。

由于需要用到多家 CDN 服务，这些企业往往需要流量在多 CDN 间进行调度与管理，在这种背景下，流量管理系统（Traffic Management System，TMS）应运而生。TMS 可以根据用户的定制需求，选择特定 CDN 服务商的最佳 CDN 边缘节点服务用户，从而更好地满足用户的网络访问加速需求。

8.4.1　多 CDN 租用调度系统（TMS）

TMS 是一种基于 DNS 的全球负载均衡（GSLB）服务，是一种增强的 DNS 服务，核心由定制的域名服务器和带有扩展的高级标准 DNS 功能的监控代理组成。支持透明地把流量路由调度到任何在公网上的 CDN（包括第三方 CDN）或服务器上，在 CDN（或服务器）间进行流量分布调度。TMS 可以独立部署，也可以与第三方 CDN 进行协同服务。

目前许多 CDN 服务商提供 TMS 的流量调度管理功能，但从灵活调度的角度出发，互联网企业自行建设 TMS 是较好的选择。

TMS 是以 DNS 服务为核心，围绕用户调度策略和调度服务为主体而设计的，如图 8-8 所示，其主要功能模块包括 DNS 服务、策略管理、资源管理、用户管理等，为了便于管理，通常包含 TMS 管理界面[10]。

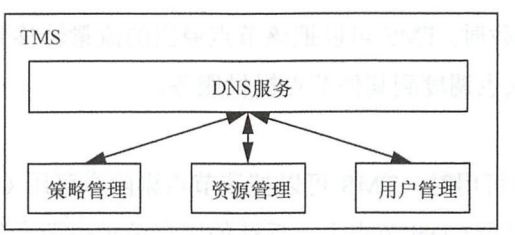


图 8-8　TMS 架构

（1）DNS 服务

这是 TMS 的核心功能，包括 DNS 域名服务器的功能以及扩展的高级标准 DNS 功能。DNS 服务可以接收用户 DNS 访问请求，结合配置的路由调度策略进行解析，向用户返回解析出的 IP 地址结果，将用户请求调度到最佳的 CDN（或服务器）上[9]。

（2）调度策略管理

需要对所有的路由调度策略进行管理，可以配置不同的策略实现根据不同的路由调度策略来调度用户的请求，目前至少包括地域策略、全球负载共享策略、溢出和内容分发对等策略、失效备援策略，下面介绍这些调度策略[10]。

• 地域策略

地域策略可以根据终端用户的IP地址所在地区，将用户调度到在该地区具有优势的CDN节点（比根据用户所用的本地 DNS 的 IP 地址调度，更为准确）。TMS 需要支持基于终端用户的 IP 地址的域名解析，实现从 IP 地址到地理区域的所有第三方映射。可以实现基于洲、国家、州（省份）以及地市级别的域名解析策略。例如，对于欧美地区用户，调度到 Level3 CDN；对于亚太地区用户，调度到 SwiftServe CDN。

TMS 需要支持 EDNS 协议（Google 提交的扩展 DNS 协议，允许在 DNS 请求中除了本地 DNS 地址，还增加用户的 IP 地址），对于使用公共 DNS（如 Google8.8.8.8）的用户，也可调度到用户真实所在地区。

• 全局负载共享策略

全局负载共享策略，是根据各个 CDN（或服务器）节点的负载能力，确定流量比例的分配，对于终端用户的请求，TMS 按照既定的比例将请求调度到各个 CDN 节点。

例如，有 3 个 CDN 节点，TMS 根据 CDN 节点负载能力，确定流量比例分配为 CDN1：60%，CDN2：30%，CDN3：10%，并且按照此比例来确定用户请求的调度。

• 溢出和内容分发对等策略

当某个 CDN 节点超载时，TMS 可以把该节点溢出的流量转移到备用服务器或其他 CDN 节点上，后续的用户请求也调度到其他节点提供服务。

• 失效备援策略

当某个 CDN 节点不可用时，TMS 可以把该节点剔除在可用 CDN 队列之外，同时把该节点的流量转移到其他可用 CDN 节点上，后续的用户请求也调度到其他可用 CDN 节点。

配置调度策略类似于生成一棵策略决策树。该策略决策树的叶子（资源节点）上是 TMS

对 DNS 请求的响应结果（IP 地址或 CNAME），分支节点上是各种决策准则。

其中，叶子（资源）节点包括 IP 地址、CNAME 记录、负载共享（流量比例）以及 DNS 记录（DNS 授权记录）。而分支节点是由地理区域（洲、国家、省份、城市）构成的。TMS 在选择最终用户调度的结果中，依据策略决策树决定。首先根据用户请求所在地理区域选择分支节点。然后根据叶子节点，选择并返回 DNS 响应结果。

例如，美国用户访问域名 www.abc.com，策略决策树配置如图 8-9 所示。

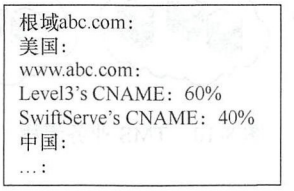


图 8-9　策略决策树配置

图 8-9 所示的决策树配置表示，在根域 abc.com 中，首先找到分支节点美国，再找到节点 www.abc.com。然后，计算用户请求的概率，60%的概率返回 Level3's CNAME，40%的概率返回 SwiftServe's CNAME。

（1）资源管理

TMS 可以管理各个 CDN（或服务器）节点的状态，通过监控或负载反馈 URL，关注各节点的流量、用户数、可用性、错误状态及原因等，并实时上报给 DNS 服务模块更新。

（2）用户管理

对用户访问请求的管理，可以保存和查看历史用户访问请求，并提供路由调度信息的统计数据，以便后续做进一步的分析。用户访问请求记录包括 IP 地址、CNAME 记录、负载共享服务记录、DNS 记录。用户通过前台页面使用用户管理功能，包括查看历史用户访问请求，根据不同分类来筛选查看历史用户访问请求，导出用户请求统计报表等。具体历史用户访问请求信息由数据库或者日志文件获得。

8.4.2　服务调度流程

从总体上看，TMS 业务流程包括用户请求、本地 DNS 解析、TMS 调度以及请求 CDN 服务。以广州地区用户为例，具体的多 CDN 租用调度（TMS）业务流程如图 8-10 所示。

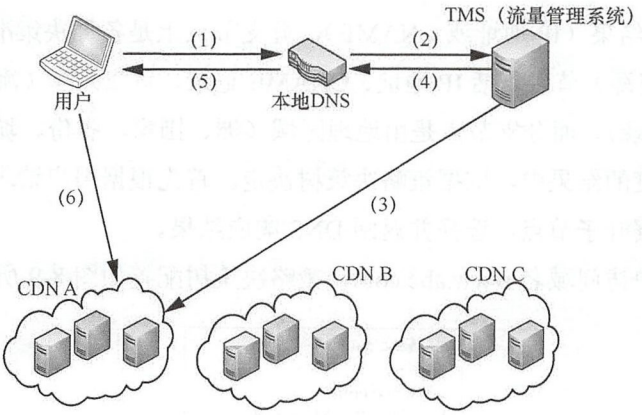


图 8-10　TMS 业务流程

（1）广州地区用户通过浏览器向本地 DNS 服务器发出访问请求；

（2）本地 DNS 服务器把域名解析权请求交给 TMS；

（3）TMS 收到请求，根据路由调度策略查询，选择最优节点 CDN A（假设有 3 个 CDN 节点，并将 TMS 的地域路由调度策略设置为中国用户调度到 CDN A，北美地区用户调度到 CDN B，其他用户默认调度到 CDN C）；

（4）TMS 将查询到的最优节点 CDN A 的 IP 地址（或 CNAME）返回给本地 DNS 服务器；

（5）本地 DNS 将该 IP 地址（或 CNAME）返回给广州地区用户；

（6）广州地区用户用浏览器访问该 IP 地址（或浏览器请求本地 DNS 继续解析 CNAME），成功访问 CDN 节点并获得请求的内容。

第 9 章

自建 CDN 实施的考虑与评估

9.1 自建 CDN 案例分析

随着互联网应用规模的不断扩大，CDN 服务在互联网上的应用也变得越来越重要。在日常的网络活动中，在网络游戏、视频直播以及线上支付等互联网应用背后，都是由 CDN 进行业务支撑的。出于更好地提高用户体验（QoE）、降低服务价格的考虑，许多大型互联网公司也会根据自己的业务背景和业务需求搭建具有不同特色的 CDN。下面对国内外的自建 CDN 使用的架构和技术进行具体分析。

（1）阿里巴巴

阿里巴巴作为中国目前大型互联网公司之一，其购物网站——淘宝每天接收到的用户请求数量是十分巨大的，因此，CDN 加速服务对于它来说是必不可少的需求。在早期，阿里巴巴还没有开始自建 CDN 时，一直使用的都是适用于通用性网站的商用 CDN 服务。但是这种商用 CDN 对于以商品小图片为主的内容分发没有什么优势，特别是在各种节日活动，例如“双十一”购物节这种高并发请求的业务场景下，常常因为磁盘 I/O 过高导致宕机，严重影响服务。为了解决这种问题，阿里巴巴开始自建 CDN。阿里巴巴的自建 CDN 早期主要针对的是淘宝购物业务，所以又被称作淘宝 CDN。后来随着云计算技术的发展，阿里巴巴将 CDN

与云结合，变成了云的一部分，即阿里云。其支撑的业务也由早期的网购业务变得多样化，从图片加速到视频加速，从小文件分发到大文件分发，阿里 CDN 的能力越来越强大。

阿里 CDN 能力的强大离不开它 CDN 的设计，主要分为缓存节点（一级缓存节点、二级缓存节点等）、内容库、DNS 服务器以及全局调度系统等。阿里云中 CDN 的组件分层为服务层、应用层、系统层和设备层。服务层主要提供流媒体分发、大文件分发、应用加速、日志分析等功能；应用层是最核心的功能实现的地方，包括全局负载均衡、本地负载均衡、缓存、监控等功能实现；系统层则是操作系统部分，阿里云主流操作系统是 CentOS；设备层提供服务器、路由器、交换机等设备管理。

本书将重点介绍阿里 CDN 的应用层部分，对于 CDN 的应用层而言，阿里巴巴的自建 CDN 利用开源软件搭建了自身的缓存节点，采用四层和七层负载均衡的策略实现节点内本地负载均衡。阿里巴巴利用 LVS（Linux 虚拟服务器）实现四层负载均衡。LVS 作为阿里巴巴自研的软件，但是目前已经开源且应用广泛，此处也将列为开源软件介绍。LVS 采用 DR 模式进行工作，采用轮询进行负载均衡的调度，双 LVS 做主备，中间有心跳监测。另外，采用 Tengine 实现七层负载均衡，Tengine 是阿里巴巴基于 Nginx 开发出来的一款 HTTP 服务器，底层也是通过调用 Nginx 的模块来实现请求调度的。Tengine 通过采用一致性 Hash 技术提高 CDN 节点命中率。Swift 作为阿里 CDN 的缓存服务器，可以支持 HTTP 以及实现内存存储、磁盘存储等。并且，阿里 CDN 也使用了 Swift HTTP 处理引擎、回源、存储以及索引等核心组件。值得一提的是，阿里巴巴根据 Swift 的 COSS 文件存储系统开发了 TCOSS 文件存储系统，降低了 I/O 读写率。

从整体上看，阿里自建 CDN 主要采用目前 CDN 最热门的软件组合——LVS+Nginx+Swift 来实现基础的缓存加速功能，其技术架构也是大多数企业所使用的，以下的各典型案例中将不再过多介绍与阿里巴巴重复的技术部分，将重点介绍每个企业 CDN 的特点。

（2）腾讯

腾讯类似于阿里巴巴，在面对大量用户的请求时也不得不自建 CDN 以提供稳定可靠的业务服务。腾讯目前也将 CDN 与云结合，称为腾讯云。从目前掌握的情况上看，腾讯在国内的 800 多个自建 CDN 节点覆盖了中国电信、中国移动、中国联通三大运营商，还在海外建设了 100 多个节点，从而能够保证全球范围内的加速。

对于腾讯自建 CDN，也包括内容库、缓存模块、DNS 服务等调度模块。值得一提的是腾讯自研的全局调度系统，简称 GSLB，GSLB 是在开源软件的基础上进行自行研发，可以

分析海量的节点数据，并对用户的请求精准调度到最优的节点，同时，腾讯的 GSLB 团队推出了一种全新的 HttpDNS 域名解析调度系统。HttpDNS 的特点在于将原来域名解析的协议 DNS 简单地换成了 HTTP，但这一微小的转变，却产生了两方面优势：一方面，由于不再经过本地 DNS 解析，而是通过 HTTP 将用户的域名解析权直接交给 HttpDNS 服务器，从而避免域名解析异常等问题造成的困扰；另一方面，区别于传统的基于 DNS 的全局调度，HttpDNS 服务器能直接获取用户的 IP 地址，实现了更精准的基于用户 IP 地址的调度。

（3）中国电信

近年来，随着 OTT、手机视频和云服务的发展，同时随着 4G 网络的发展与成熟，未来视频业务迎来爆发性的增长，国内的运营商为了应对业务的发展也纷纷开始搭建 CDN。对于国内的三大运营商（中国电信、中国移动、中国联通）而言，CDN 业务主要包括流媒体加速、网页加速以及下载加速等，这些业务对应的 CDN 工作原理和技术架构在 3 个运营商中大体相同，但也会有一些差异。本书中将以中国电信的 CDN 作为运营商建设 CDN 的例子进行分析，分析在运营商业务背景下的 CDN 相关技术。

中国电信的整个 CDN 可以分为全局缓存节点、内容路由系统以及内容管理系统等。内容管理系统主要负责整个 CDN 系统的管理，包括内容注入、内容分发、内容发布等。目前中国电信采用两种主流的内容分发技术：Push（主动分发）和 Pull（被动分发）。一般是热点资源采用 Push 的方式发送到边缘，较冷的资源通过缓存节点向上级内容源拉取获取。内容路由系统负责将用户的请求调度到合适的设备，内容路由可以划分为全局负载均衡（GSLB）和本地负载均衡（SLB）两类。GSLB 主要包括基于 DNS 和基于 HTTP 两类重定向，通过用户请求分流到离用户最近的地区；SLB 是将用户请求进一步调到提供服务的流媒体服务器上。

目前，中国电信在北京、上海、广州都建设了 CDN 的内容中心，同时在浙江、江苏、福建等地建立了 CDN 的省级骨干节点，保证了并发服务能力。同时，在统一 CDN 管理支撑系统之下，垂直建立多个分发加速子网络，对外包装为统一的业务分发能力和资源。

（4）Netflix

作为全球领先的美国互联网电视网络公司，目前 Netflix 拥有来自 50 个国家的 7 000 多万个用户。Netflix 首先将片源发送到自建 CDN 上，然后把内容推送给本地的互联网服务提供商。Netflix 选择自建 CDN 主要是由于随着流媒体视频业务的快速增长，必须扩大其基础设施，以满足 Netflix 视频流的增长速度需求。

Netflix 的 CDN 为拥有超过 10 万个用户的较大型 ISP 提供服务。Netflix 的 CDN 系统的开放连接不仅能通过 ISP 与 Netflix 各地的数据中心进行连接，并且 Netflix 能给 ISP 免费提供 CDN 服务器。OCA 是 Netflix 的 CDN 最具有特色的地方，通过 ISP 直接访问 OCA 能够极大降低传输成本。Netflix 具有三层缓存结构，第三层是 ISP 内部的 OCA 的服务器，第二层是 Netflix 的数据中心，而最底层就是在 AWS 的 S3 服务器。Nginx 作为 Netflix CDN 的流媒体服务器，当 CDN 开始运行时，Netflix CDN 能够适时微调 Nginx 的设置。进一步来说，结合 FreeBSD 和 Nginx 的产品具有事件驱动性设计，可以避免 I/O 阻塞，并提高 CDN 服务性能。

（5）Comcast

Comcast 作为美国最大的有线电视传输和宽带通信公司，为应对流媒体视频业务的不断升温，推出了自己的 CDN 服务，目前，已在北美地区部署了大约 100 个节点。Comcast CDN 支持各类 HTTP 形式的媒体内容，主要面向时下快速发展的 OTT 业务，并且对于广播及有线电视网络商、在线内容商、游戏公司以及软件公司的不同业务和终端能够实现跨设备连接。

2015 年，Comcast 将其 CDN 的核心在首次开源发布——Apache Traffic Control（ATC）。ATC 的主要功能是通过一个控制平台把 CDN 的边缘服务节点汇聚为一个缓存节点，进而组成一个 CDN。同时，ATC 使用了 Apache Traffic Server（ATS）作为其缓存与 HTTP 服务的软件。

目前，Comcast 的 CDN 在结构上也由 CDN 流量调度、缓存、流量监控与统计、CDN 门户以及 CDN 日志模块等构成。流量调度保证了用户可以访问到最优的缓存节点以获取想要的内容，可以配置基于 DNS 调度或者基于 HTTP 重定向调度测策略；缓存节点为用户加速了内容的访问，并选用了 ATS 技术进行搭建；管理监控系统能够监控各个节点，保证 CDN 的稳定运行；日志系统可以获取各个临界点的日志，以此来分析各个节点的情况。

9.2 自建 CDN 中的开源软件

从第 9.1 节中的国内外各大企业的自建 CDN 案例可以看出，目前 CDN 可以划分为内容路由系统、内容管理系统、缓存节点、管理监控系统。每个部分涉及的开源软件在第 9.1 节中也有提到，不同公司可能会采用不同的软件实现相同的功能，本节中将对每个部分涉及的主流软件进行介绍与对比，为自建 CDN 提供软件选用方面的参照。

9.2.1　缓存系统

关于缓存节点，其主要功能就是对用户资源进行缓存，以此提高用户的访问请求速度。缓存节点是 CDN 整个网络中十分重要的部分，本节将详细介绍缓存节点常用的开源软件，见表 9-1 [11]。

表 9-1　缓存软件框架性能和功能对比

软件	Squid	Nginx	ATS	Varnish
优点	• 具有磁盘缓存容量优势； • 支持 ACL 角色控制，支持 ICP 缓存协议； • 支持外部规则文件读取及热加载	• 多核支持，支持代理插件； • 通过插件可以充当多角色服务器	• 支持多核，支持磁盘/内存缓存； • 支持插件开发，支持 ICP； • 支持外部规则文件读取及热加载	支持多核，支持内存缓存
缺点	不支持多核	不支持外部文件读取，需要转义	系统复杂性较高，配置困难	• 无法磁盘缓存； • 不支持集群，支持后端存活检查； • 不支持外部文件读取，需要转义
总结	磁盘缓存容量优势，支持多种协议规则，无法多核	支持多核，功能多，但读取外部文件麻烦	功能与性能均非常强大，但配置较复杂	无法磁盘缓存、无法集群，文件读取麻烦，不利于管理

9.2.2　内容管理系统

关于内容管理系统，其主要功能就是内容资源的注入、存储以及分发。目前可以采用以下几款软件实现，见表 9-2。

表 9-2　内容管理系统框架性能和功能对比

软件	HDFS	Ceph	Lustre	GoogleFS
优点	• 支持大数据的批量读写； • 可以支持按顺序读写，一次写入，多次读出	• 开源分布式文件系统； • 支持 POSIX； • 操作方便	• 开源； • 支持 POSIX； • 文件被分割成 Chunk	成本低，可以运行在廉价的普通硬件上

（续表）

软件	HDFS	Ceph	Lustre	GoogleFS
缺点	• 交互式应用，低时延难满足； • 不支持多用户并发响应文件	存在单点故障	没有元数据服务器，增加了客户端的负载，占用了相当的CPU和内存	不开源
总结	若是很多小文件，处理压力大	实施简单但会存在单点故障	适用于大文件，当用于遍历时实现复杂	不开源使用困难

9.2.3 内容路由系统

关于内容路由系统，其主要功能就是对用户的请求进行调度，调度到距离用户最近的区域或者分流到最优的节点。目前主要采用以下几款开源软件实现，见表 9-3。

表 9-3 内容路由系统框架性能和功能对比

软件	LVS	Nginx	Haproxy
优点	• 抗负载能力强； • 工作在网络第四层； • 支持 RR（轮询）、Weighted-RR（加权轮询）负载均衡算法	• 工作在 OSI 模型的第七层； • 可以承受高的负载压力； • 可以支持多种负载均衡算法； • 可以做 Web 服务器	• 支持两种代理模式：TCP（四层）和 HTTP； • 支持多种负载均衡算法
缺点	• 不支持正则处理，不能做动静分离； • 配置复杂，对网络依赖比较大	不支持会话的直接保持	无法做 Web 服务器
总结	运行在四层上，稳定性高，转发效率高，但是 LVS 对应的维护人员要求也高	支持七层负载均衡，用户量最大，稳定性较高，不支持会话保存	支持四层、七层负载均衡，支持更多的负载均衡的策略，支持会话保存

9.2.4 监控系统

关于管理监控系统，其主要功能就是对 CDN 中的各个节点进行监控，获取每个节点的健康状况，以此保证 CDN 能够正常地工作。目前可以采用以下几款开源软件实现，见表 9-4。

表 9-4　监控系统框架性能和功能对比

软件	Cacti	Nagios	Zabbix
优点	• 提供 Web 操作； • 手动建立监控方式； • 集中式的监控方式	• 文件编辑操作，没有 Web 操作； • 提供集中式、分布式的监控方式	• 提供 Web 操作； • 自动建立或者手动建立监控连接； • 提供分布式、集中式的监控方式
缺点	不支持表达式	不支持表达式	监控图像功能比较简单
总结	操作灵活，监控方式单一	操作方式不灵活，集中、分布式监控方式均支持	易于操作并建立连接，集中、分布式监控方式均支持

不同的开源软件各有利弊，目前，搭建 CDN 已被广泛应用的开源软件组合如下。

- 内容管理系统：Ceph；
- 缓存：LVS+Nginx+Squid；
- 全局内容路由系统：智能 DNS 调度和 HTTP 应用层调度；
- 监控管理系统：Zabbix+Grafana。

值得注意的是，企业在选用开源软件的时候，需要根据自身业务需求进行考虑与选择。

9.3　自建 CDN 部署

CDN 的作用在于解决内容长距离传输和跨域传输带来的种种时延和业务质量问题。因此，为用户提供服务的节点一定要部署在靠近用户的地方。但从 CDN 的建设成本上考虑，把所有内容全部存放在这些离用户最近的节点中会消耗大量存储成本，所以，这些提供服务的节点会根据需要从源站服务器或者其他上级缓存服务器拉取内容[3]。这样一来，就形成了 CDN 架构分层部署的需求。

从部署方式上看，CDN 可以分为分布式部署和集中部署两种方式。正如第 9.1 节所述的例子，CDN 由内容管理系统、调度系统、缓存系统、监控管理系统等部分构成。针对不同部分，可以采用不同的部署方式。例如，内容管理系统可以采用分布部署对外提供分布式文件系统，并行处理文件的读写；调度系统采用集群部署方式，提供统一调度处理功能等。CDN 的各个模块可以根据功能、性能的不同要求采用不同的部署方式。

从架构层级上看，互联网 CDN 通常是中心和边缘两级架构，而流媒体 CDN 往往是中心、区域、边缘三级架构。这种区别产生的原因是网站内容的大小通常都是固定的，并且在整个

文件完全被获取后，才展示给用户，用户几乎感觉不到在传输过程中的分组丢失、抖动等。但流媒体在线播放的过程中，如果出现这样的问题，用户就会看到花屏、卡顿，感觉非常明显[3]。因此，媒体对带宽需求很高，边缘节点向源站或中心节点回源拉取内容，要比互联网CDN回源消耗更多资源。特别是对于流媒体直播，只要直播没有结束，服务器就需要持续不断地发流，要是没有第二级区域节点作为中继，那么源站和中心节点是无法承受如此大的回源压力的[3]。流媒体CDN的三级部署方式如图9-1所示。

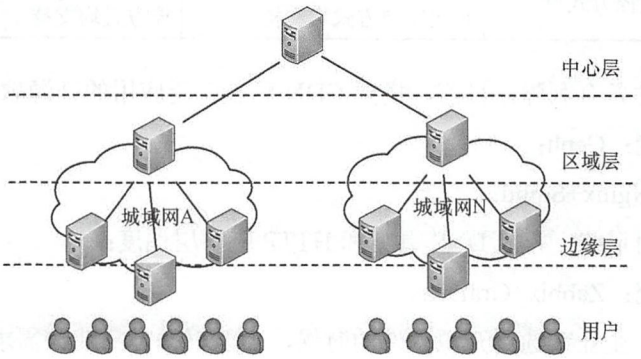


图9-1　流媒体CDN的分层部署方式

9.4　自建CDN技术指标

在第1章中介绍了互联网应用的多种服务形式，如Web应用、流媒体、动态网页、文件下载等，为了保证用户服务质量（Quality of Service，QoS）和用户体验质量（Quality of Experience，QoE），对CDN的服务指标有着严格的要求，具体介绍一些常见的自建CDN服务指标。

以视频业务为例，服务监控应重点关注视频的QoS相关参数，包括HTTP响应时延、视频打开时延、视频下载速率等参数。视频质量QoS与QoE关系如图9-2所示。

9.4.1　命中率

在前文中已经提到，CDN的主要功能就是将网络的热点内容缓存到本地节点，用户通过调度机制访问最近的CDN服务节点，如果被访问的内容已缓存，则直接由该节点提供服务，

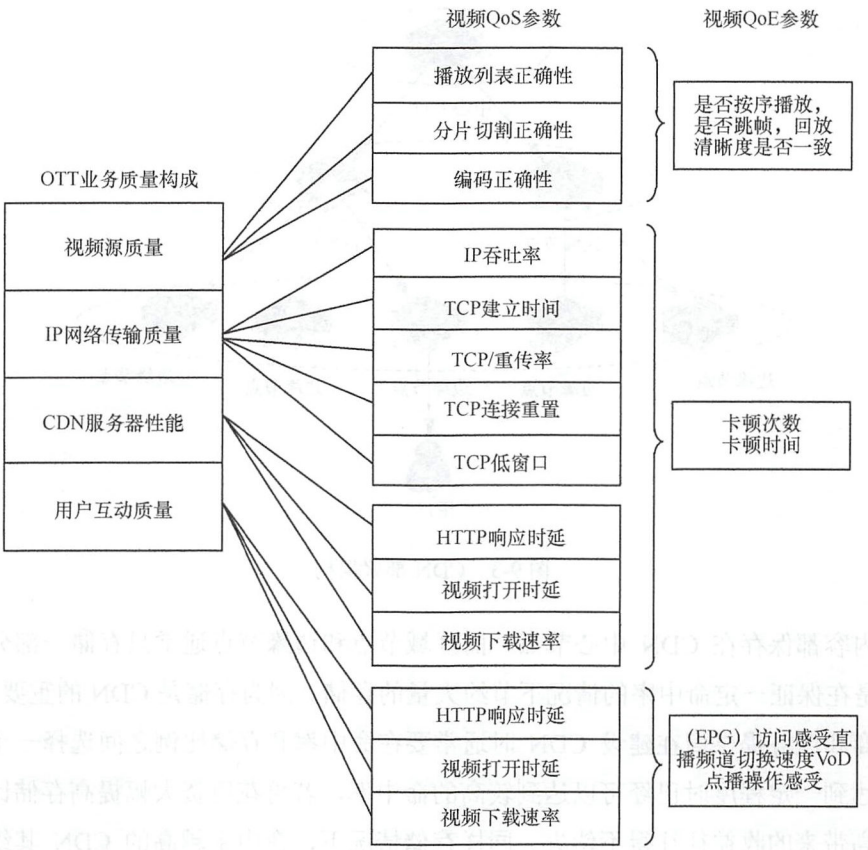


图 9-2　视频 QoS 与 QoE 参数

否则用户被重定向至源地址进行访问。CDN 的效率体现在用户的访问内容是否在缓存中，即“命中率=命中次数/（命中次数+未命中次数）”。假设用户通过 CDN 访问某网站内容，命中次数是 a，未命中次数是 b，可计算出命中率为 $a/(a+b)$，未命中率为 $b/(a+b)$。可见命中率越高，说明 CDN 效率越高，能够尽可能通过本地缓存满足用户的访问请求，从而减轻源地址的访问压力，优化网络流量[1]。

下面结合 CDN 来详细说明命中率的意义，图 9-3 所示为一个典型的 CDN 部署架构。

CDN 从网络结构上看，通常分为三层：中心节点、区域节点和边缘节点。中心节点将内容推送到区域节点，区域节点再将内容推送到边缘节点，由边缘节点为用户提供服务。边缘节点未命中时，会将用户重定向到上一级的区域节点，若区域节点仍未命中，则将用户调度到上一级的中心节点[1]。

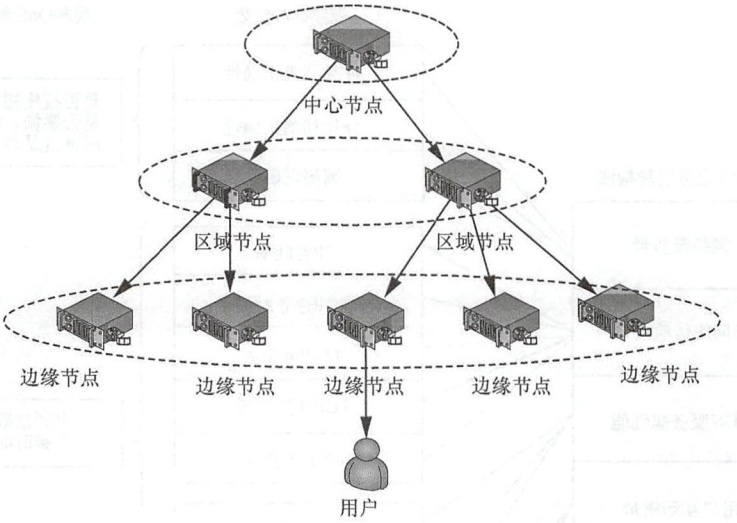


图 9-3　CDN 部署架构

所有内容都保存在 CDN 中心节点，而区域节点和边缘节点通常只存储一部分热门的内容，目的是在保证一定命中率的情况下节约大量的存储。因为存储是 CDN 的重要成本之一，网络运营商和服务提供商在建设 CDN 时通常要在命中率和存储比例之间选择一个平衡点，存储比例达到一定程度时已经可以达到较高的命中率，若再花巨资大幅提高存储比例，命中率小幅提高带来的收益往往得不偿失。同样存储情况下，命中率越高的 CDN 其缓存技术实现也越好，在满足需求的情况下越能节省建设成本。

在运营商 IPTV CDN 的实际配置模型中，每个区域节点通常存储 30%~50%的热点节目内容，每个边缘节点通常存储 20%~30%的热点节目内容，CDN 中心节点会不断统计每个节目内容被点播的次数，达到一定阈值即判断为热点内容并推送到区域节点和边缘节点。边缘节点的存储比例已经能够满足大部分用户的访问请求，若边缘节点未能命中内容，则将用户调度到上一级的区域节点进行服务，只有非常少量的冷门内容请求会被调度到中心节点进行服务[1]。

9.4.2　吞吐量

CDN 的吞吐量通常指的是单台 CDN 服务器在单位时间（一般以秒计算）内成功传输的数据大小（一般以 byte、KB、MB、GB 计算），是 CDN 服务器的重要性能指标之一。CDN 服务器可分为机架服务器、刀片服务器、机柜式服务器等多种形态，其吞吐能力取决于多方

面因素[1]，具体如下。

- 硬件配置：CPU、内存、磁盘类型、磁盘大小、磁盘缓存、网卡类型、网卡速度、总线速度等，其中硬盘 I/O 通常是瓶颈。
- 软件配置：操作系统版本、管理软件等。
- 网络带宽：取决于 CDN 服务器在网络中的位置，运营商的 CDN 服务器通常放置在城域网出口等关键位置，能够保证带宽需求。

表 9-5 给出了 CDN 服务器的一些关键参数，对吞吐量有较大影响。

表 9-5　CDN 服务器关键参数

参数	描述
CPU	主要为 Intel 和 AMD 的服务器级多核 CPU，如 Intel XEON 系列、AMD 皓龙系列等
内存	以 DDR4 为主，大小 128~256 GB
磁盘	SATA/SAS/SSD，大小 1~8 TB
网口	分为电口、光口，其中电口通常为吉比特以太网口，光口可支持更高速率
操作系统和软件	Windows 服务器/Linux，32 位/64 位，CDN 管理软件

9.4.3　并发值

并发值也是 CDN 的重要服务指标之一。从 CDN 服务类型看，有两种服务类型，首先是 Web 缓存、动态网页、文件下载等互联网应用服务，这类服务通常传输数据量较小，单位时间内频繁发起 TCP/UDP 连接，对 CDN 的会话处理能力要求很高；其次是视频点播、视频直播等流媒体服务，这类服务通常要求较高的带宽以保证视频稳定传输，TCP/UDP 连接请求不如前者频繁，但对 CDN 的吞吐能力要求很高[1]。两者对 CDN 的并发处理能力要求见表 9-6。

表 9-6　CDN 服务类型和并发能力要求

CDN 服务类型	带宽要求	会话要求	硬件要求	并发值
Web 缓存、动态网页、文件下载等互联网应用	中等	很高	对 CDN 服务器 CPU、内存、软件要求很高	单台通常在 2 000~10 000
视频点播、视频直播等流媒体服务	很高	中等	对 CDN 服务器存储、网口、总线 I/O 要求很高	单台通常在 300~2 000

对应于并发值，还有个指标即“并发比”，指的是同一时刻发起服务请求的用户（即活跃用户）占所有用户的比例。用户的访问行为具备分散性和集中性，同一天内不同时段活跃用户的比例不一，而且出现热门内容时活跃用户的比例也和平时会有区别，图 9-4 是某省 IPTV CDN 的一个中等规模的边缘节点在一天内不同时刻的活跃用户统计。

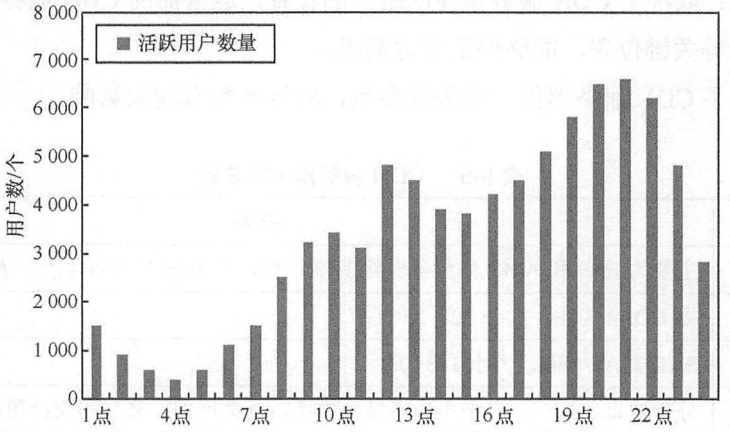


图 9-4　IPTV 活跃用户数量统计

从图 9-4 中可见，用户的活跃程度和作息时间有密切的关系，晚上 8 点到 10 点是用户访问的高峰期，活跃用户数量达到了当天的最大值。网络运营商和服务提供商在建设 CDN 时通常要考虑各个节点的并发值，避免建设能力过多造成投资浪费，在 IPTV 业务中，运营商根据以往运营的经验值测算并发比，该值通常为 1:3 左右，即如果该节点服务的 IPTV 用户总数为 10 000，只要按照 3 000 左右的并发值进行 CDN 建设即可满足用户需求。同理，建设区域节点和中心节点时，也要充分分析该节点的并发值需求，合理地进行建设[1]。

9.4.4　响应时间

在 CDN 中，响应时间是指用户通过终端的浏览器或者客户端发出内容访问请求，到收到内容第一个数据分组的时间，也可称为端到端响应时间。响应时间对用户体验有很大的影响，响应时间过长会造成用户心理烦躁和不安，逐渐失去使用该业务的兴趣，因此响应时间也是 CDN 最重要的服务指标之一。一个完整的端到端响应时间通常由图 9-5 中的几部分组成[1]。

图 9-5　端到端响应时间

- DNS 地址解析：用户终端将内容请求的域名地址发送给本地 DNS 进行解析，返回 CDN IP 地址。
- 访问 CDN 服务器：用户对 CDN 的访问请求通常会被指向 CDN 的负载均衡设备进行处理。
- CDN 内部重定向：CDN 负载均衡设备根据用户 IP 地址、节点负载情况、网络流量等综合信息，将用户重定向到离其最近的 CDN 服务节点。
- 收到数据分组：用户终端与 CDN 服务节点建立连接，收到 CDN 服务节点发来的第一个数据分组。

以上各部分精确的响应时间可以通过抓取分组软件或第三方专用仪表测量。目前网络运营商和服务提供商对响应时间都有严格的要求，以 IPTV 业务为例，如节目菜单（Electronic Program Guide，EPG）响应时间不超过 2 s，流媒体点播、直播、快进/快退/暂停/定位等操作响应时间不超过 1 s，频道切换时间不超过 1 s 等，这些要求可以充分保证用户的服务质量和业务体验。

9.4.5　MDI

媒体传输质量指标（Media Delivery Index，MDI）是由思科和 IneoQuest 公司联合提出的一套媒体服务指标，通常用来测量和评估基于 IP 网络的视频流媒体的传输质量。MDI 并不限定视频流媒体的传输格式和编码格式，这使得它可以广泛应用于 IPTV、OTT、数字电视等基于 IP 网络的流媒体领域。RFC 组织于 2006 年发布了 MDI 相关标准（RFC4445），并已成为 IP 视频流媒体传输质量测试的行业标准。

MDI 是媒体延迟和媒体丢失率的综合评价指标，包括两个重要参数[1]。

（1）DF（Delay Factor，延迟因子）

DF 值用来衡量视频流的延迟和抖动情况，单位为毫秒。DF 是指单位时间（通常为 1 s）内被测试设备流入与流出的视频流字节数除以视频码率的最大值与最小值的差值，如式（9-1）所示：

DF=（max[流入字节数–流出字节数]–min[流入字节数–流出字节数]）/视频码率 （9-1）

其中，流入字节数即设备接收到的字节数，可通过实际测量获取，而流出字节数即视频解码所需的字节数，可通过分析视频流解码获取。

DF 值反映的是视频流的抖动变化情况，由于流媒体的实时性，用户终端一方面通过网络获取视频流，一方面通过视频解码器消耗缓存的视频流，而视频流的数据分组到达时间间隔因为网络抖动往往容易出现变化，DF 值就是将网络抖动情况换算为对视频流进行缓存的需求。当解码器缓存的视频内容大于 DF 值时，不会出现缓存内容已消耗完而没有后续的视频流这种情况，即网络抖动不影响视频播放质量；当 DF 值较大已超过解码器缓存的视频内容时，解码器消耗完了缓存内容而没有收到后续的视频流，就会出现卡顿、马赛克等现象。

因此，DF 值越小说明视频流抖动越小，对 CDN 和用户终端等设备视频缓冲区相应的要求越低，不会影响视频播放效果。

（2）MLR（Media Loss Rate，媒体分组丢失速率）

MLR 指的是单位时间（通常为 1 s）内丢失的视频流字节数，如式（9-2）所示：

MLR=（应收到字节数–实际收到字节数）/时间 （9-2）

MLR 值反映的是视频流的数据分组丢失情况，通过测量 MLR 可以及时了解视频流传输的网络质量。数据分组丢失将直接影响到视频的播放效果，因此理想的 MLR 值为 0，由于视频流在编码过程中通常会采用一些冗余和纠错技术，而且在传输过程中会采用分组丢失重传技术，因此在实际应用中可根据具体情况设定 MLR 上限。

由于 MLR 只反映单位时间内的分组丢失速率，不能反映分组丢失的持续性，MLT（Media Loss Total，媒体分组丢失总数）指标则是统计总共分组丢失数量，其中 MLT-15 用来统计之前 15 min 的分组丢失总数，MLT-24 用来统计之前 24 h 的分组丢失总数，结合这些指标可以更好地了解测试期间的媒体分组丢失情况。

9.4.6 MOS

MOS（Mean Opinion Score，平均意见指标）值早期主要用于评价通信系统中的语音质量优劣，目前 MOS 值已延伸到了视频通信等领域，通常也用来评价流媒体类型 CDN 的服务质量。MOS 值的取值区间是[0, 5]，分数越高说明视频质量越好，MOS 值各级别见表 9-7。

表 9-7　MOS 值对照

MOS 值	级别	用户满意程度
5.0	优秀	非常好，画面清晰，声音连续，无马赛克、停顿等现象
4.0	良好	还可以，画面较清晰，有些延迟，有些杂音
3.0	中等	一般，有较多马赛克、花屏现象，经常停顿
2.0	较差	勉强，频繁出现马赛克、停顿、杂音等现象
1.0	很差	极差，完全看不清画面，听不清声音

MOS 值测量方法有两种：一种是主观评价，另一种是客观评价[1]。

主观评价方法是选取一组测试评价人员（通常不少于 10 个），由他们分别观看视频源文件和通过用户终端观看视频内容并进行对比，每个人都基于观看效果进行主观打分，然后将各位人员的分数进行平均计算得出主观 MOS 值。

客观评价方法则是通过第三方测试仪表或软件进行测量，目前如思博伦、安捷伦等专业仪表厂商的相关产品均可测试 MOS 值。

9.4.7　稳定性和可靠性

由于 CDN 设备需要长时间为用户提供 Web 缓存、文件下载、流媒体等内容的分发服务，这些媒体服务具备并发量大、传输码流高、连接时间长等特点，对 CDN 设备的稳定性和可靠性要求很高，以下是一些常见的 CDN 相关服务指标要求[1]：

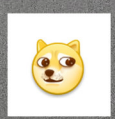
- 用户点播、直播流媒体服务的成功率不低于 99%；
- 用户请求、切换页面成功率不低于 99%；
- 用户访问内容准确率不低于 99.9%；
- 单次流媒体服务故障率不超过 0.1%；
- 单个 CDN 节点故障率不超过 0.01%；
- 当 CDN 节点出现故障时，将用户切换到另一个节点的时间不超过 5 s；
- CDN 设备存储支持热插拔技术，更换磁盘不影响用户访问请求。

由于性能的个性化需求，自建 CDN 的原因就是传统商业 CDN 无法满足业务定制的需求。自建 CDN 是自行开发和管理相应的 CDN 系统和平台，需要内容供应商的规模足够大，足以

支撑一个全网的 CDN 服务，同时具备相当的技术力量和维护力量，需求相对单一明确，复杂度不高，个性化较强。自建的 CDN 系统多以某种开源系统作为基础，针对自身个性化需求进行开发设计，从而更好地适应内容运营的需要。自建 CDN 在国内应用较多，各大门户网站和视频网站均有自建的 CDN。

第三部分

基于开源的自建CDN设计



第 10 章

开源 CDN 架构设计

10.1 业务需求

目前流媒体技术被广泛使用，很多视频的点播、直播业务以及网络广告等都应用了流媒体技术，这些给互联网提供商的服务器带来了巨大的压力。因此，在利用开源软件构建新型 CDN 网络架构时，需要考虑到流媒体业务快速发展带来的网络冲击，同时也需要考虑到互联网内容提供商最需要解决的问题。此外，CDN 的架构设计者需要分析目前 CDN 用户的需求，并与用户进行充分沟通，确认自建 CDN 能满足其访问需求。

以源站 www.testcdn.net 为例，分析互联网提供商的需求和 CDN 用户对于自建 CDN 需要解决的问题，为 CDN 用户提供服务的源站界面如图 10-1 所示。

充分与 CDN 用户沟通后，了解到源站中包含图片和视频等内容，并且源站的用户遍布全球，每天的访问流量来源是中国占 50%、欧洲占 30%、美洲占 20%。在了解 CDN 用户的基本背景后，获知 CDN 用户想通过 CDN 的服务能力帮助其实现以下几个需求。

（1）网页加速

希望能够通过 CDN 对整个网站的性能进行整体的提升，当本国甚至全球的用户在访问网页时，都可以快速获取到网页中的资源，而不会出现响应时间过长，甚至超时的现象。

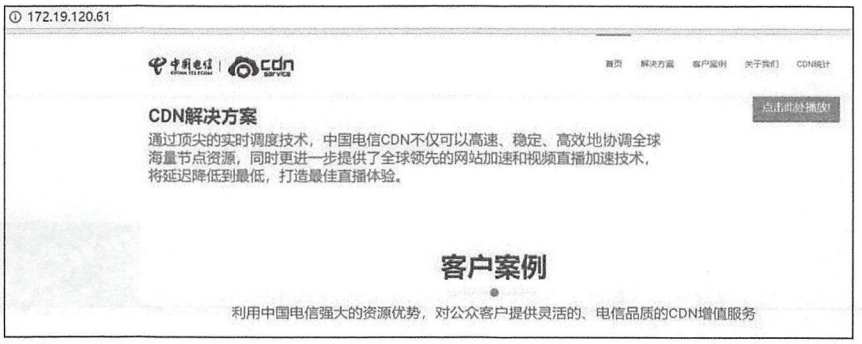


图 10-1　源站首页

其中，了解到该源站中的网页涉及图片、代码等多种元素，有些元素不经常更新，而有些却经常变化，所以，CDN 在进行网站加速时也需要考虑到如何进行静态页面加速和动态页面加速。关于 CDN 的加速服务，静态页面加速是最为基本的内容缓存服务。而对于网站中经常发生更新和变化的内容，则需要采用动态加速技术。因为对于动态内容，终端用户每次访问的内容都会不同，基本的 CDN 缓存技术无法解决，可采用差异化内容缓存、优化 TCP、动态压缩等先进技术进行加速服务。

（2）HTTP 视频服务加速

针对该源站主要提供给用户的服务是网站上的各种视频播放。随着使用这个网站的用户越来越多，此源站的服务器承受的压力越来越大，良好的视频体验以及播放质量成为 CDN 用户最为在意的问题。为此，CDN 如何为源站的视频业务进行加速成为 CDN 最应该解决的问题。由于该源站上的视频主要都是 HTTP 视频，所以 CDN 需要保证如何进行 HTTP 视频加速服务。HTTP 视频点播加速需要 CDN 利用调度系统，将视频资源推送到离用户近的 CDN 服务器上，进而提高了用户的点播速度。同时 HTTP 视频点播加速也需要自主地感知用户的访问喜好，并对用户的访问请求进行分析，进而改进视频内容分布策略，为用户提供优良的视频观看体验。对于 HTTP 视频直播加速，直播流会先传输给 CDN 中的上级节点，然后上级节点将内容下发给下级节点，最后传输到用户端。

（3）智能全局调度与本地负载均衡

该源站希望 CDN 在提供基本业务的时候，还要能够保证提供最高性能的视频服务。因此，自建 CDN 需要设计智能的全局调度策略，尽可能地把用户的请求分配到距离用户最近的 CDN 节点上。对于任意 CDN 节点，内部涉及多样的本地负载均衡策略，例如，最小连接

数、服务器最小负载策略等，这些本地负载均衡策略使得 CDN 能够更加智能地应对平衡节点内部的流量增加，也能更加智能地为用户提供服务。

10.2　开源 CDN 总体架构

根据业务需求分析得知，需要为 CDN 节点建立不同的模块来实现不同的功能。然而，在针对业务需求构造出 CDN 的各个功能模块的同时，也要考虑到内容传输带来的时间消耗问题。

对于视频业务的加速而言，为了实现加速功能，需要将内容存储在 CDN 中的内容存储节点（内容库）中，内容存储设备会将存储的内容数据下发到各地的专用流媒体服务设备中，用于为用户提供服务。因此，通过流媒体节点协同内容库能够向用户提供稳定可靠的流媒体服务。

但是，一个视频从源站出发，经过内容存储中心以及各级节点最终到达各地用户时会消耗很长的时间，而用户是不希望等待这么长的时间的。因此，自建 CDN 在选择开源软件和设计架构时需要注意内容传输的时延问题。

在分析了业务需求和传输时延情况后，对自建 CDN 提出了如图 10-2 所示的架构设计。

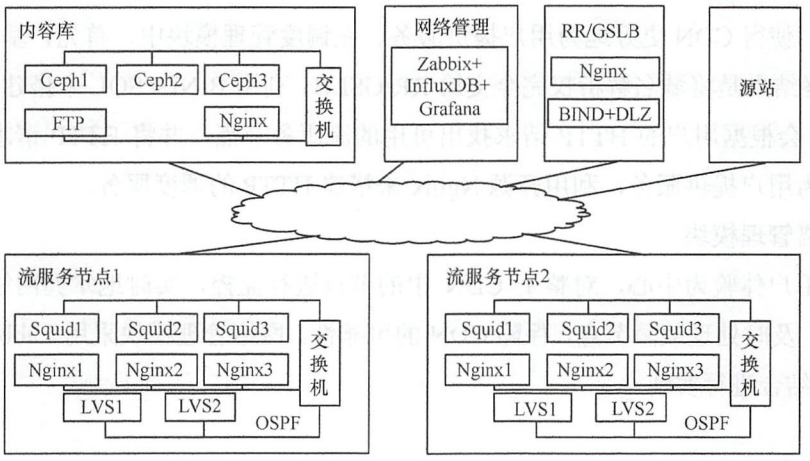


图 10-2　开源 CDN 总体架构示意

从整体来看，图 10-2 设计的 CDN 为内容库与流服务节点二级架构。具体来说，设计的自建 CDN 可以划分为内容库模块、流服务缓存模块、全局调度 RR 模块以及网络管理模块，

各个模块的功能如下。

（1）内容库模块

作为整个自建 CDN 中的核心部分，提供的功能是对内容资源进行一系列的操作，如存储并管理内容资源、将内容分发给下级节点等，并提供内容在 CDN 中的多副本分布式存储，从而实现系统存储资源、计算资源以及宽带资源的合理利用，使得用户获得更加良好的体验。内容库模块利用开源 FTP 实现内容注入，利用分布式文件存储系统 Ceph 来实现海量数据的存储以及利用 Nginx 来实现 HTTP 下载内容分发功能。

（2）流服务缓存模块

作为 CDN 中直接为用户提供流服务的模块，在面对用户请求时，将先在本地查找用户请求的内容，当本地没有命中时，则将请求转发到上级节点，并一边从内容服务器获取资源，一边对外提供服务。其中，缓存功能分担了内容库的压力，并加速了服务。采用多种开源软件相结合来设计流服务缓存模块。其中，Squid 用来实现高速缓存，OSPF+LVS+Keepalived 和 Nginx+Lua 协同实现本地负载均衡。

（3）全局用户请求调度模块

RR 主要用于完成用户的请求访问调度，为用户分配合适的流媒体节点提供服务。RR 会根据 CDN 的网络拓扑结构、各个节点的负载情况等，将用户的视频请求重定向到最适合的流媒体节点，使得 CDN 更好地为用户提供服务。在调度管理模块中，首先，基于 DNS 层面的调度，最终结果是将域名解析权完全交给 RR/GSLB，利用 BIND+SQL 来搭建 DNS 调度管理系统，RR 会根据用户的 HTTP 请求找出可用的流服务节点，并将 HTTP 请求转发给该节点，该节点为用户提供服务，利用开源 Nginx 来搭建 HTTP 的调度服务。

（4）网络管理模块

能够以用户体验为中心，对整个 CDN 中的节点进行监控，实时地掌握网络中各个节点的健康状况，及时处理故障节点，保障 CDN 的可靠性。网络管理模块采用 Zabbix、InfluxDB 以及 Zabbix 结合进行实现。

10.3 CDN 网络规划

对于使用 CDN 的用户而言，其关心的是资源的使用情况，并不关心设备的具体配置情

况。对于抽象的设备功能，可以通过网络设计模型转化为更加直观的资源利用，网络设计模型有利于对业务的进一步开展。以源站 www.testcdn.net 的用户为例，针对业务需求并结合 CDN 用户的流量分布（具体的流量分布可参见第 10.1 节），对基于 CDN 的用户的分布情况进行规划设计。针对不同地区的 CDN 用户的网络规划如图 10-3 所示。

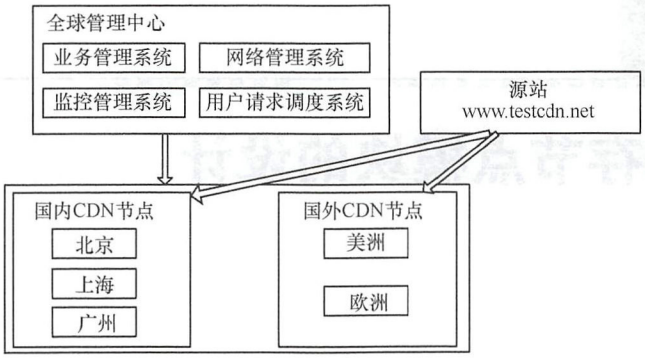


图 10-3　CDN 网络规划

从图 10-3 可以看出，通过部署国内以及国外的 CDN 节点，可以支持对各种业务的承载以及实现多种业务的加速。组网中采用统一的全球 CDN 管理中心，并按地域垂直建立多个内容分发子网络，将各地 CDN 节点的加速服务能力模块化，对外包装成一个统一高效的业务分发能力。

第 11 章

流服务缓存节点模块的设计

从上文的需求分析可知，CDN 为源站提供的最基本的服务就是图片或者视频的加速，而加速最主要的目的就是让网站的用户可以从更加靠近自己的节点处获取内容。这些靠近用户的节点在本书设计的自建 CDN 中就是流服务缓存模块。该模块作为 CDN 的边缘节点，为用户提供流媒体内容的传送与分发服务，主要由反向代理缓存以及本地负载均衡服务功能构成。以目前互联网较为常见的 HTTP 流媒体服务为例，重点介绍自建 CDN 中的流服务缓存模块的搭建。

11.1 流服务缓存节点的特性

当用户请求内容时，请求先到达边缘流服务设备。如果流服务节点没有命中请求内容，将会把请求转发到上级节点，直至向中心节点或者源站发起请求找到内容为止。由此看出，缓存的作用不仅仅在于分担内容库的压力，还可以很好地保护源站，同时也可以提高为用户就近服务的质量，对内容分发起到了加速的作用。

11.2　流服务缓存节点开源软件简介

11.2.1　Squid

Squid 常常被用作代理缓存服务器，在自建 CDN 中处于源站和客户端的中间位置，使得用户无需访问源站便可获取内容资源，提高了用户的访问速度。作为代理服务器，Squid 可以支持多种协议，如 HTTP、FTP、SSL 协议等。Squid 使用的是单独的 I/O 驱动进程来获取并响应客户端的请求，这是 Squid 独特的地方。

Squid 作为代理服务器，可以获取并响应用户的访问请求。当用户向 Squid 发出访问某个内容的请求时，Squid 会将用户请求转发到需要的网站，然后，网站响应该请求并将内容返回给 Squid，最后，Squid 将内容返回给用户，同时也会在本地存放一份备份内容，以后遇到同样的用户请求时则将备份传送给用户，以此提高用户的响应速度。

由于 Squid 存在已久，导致其与近年来流行的系统特性有很多不兼容之处。所以，目前很多公司在引用 Squid 的时候都会对其核心功能进行修改，比如，修改 Squid 以使得它支持多进程等。对 CDN 的提供服务商而言，也需要根据不同需求对 Squid 进行特定的修改。

虽然 Squid 存在时间比较长，也有很多特性无法支持，但是作为代理缓存服务器，Squid 仍然能为用户访问网站起到很好的加速作用，并且在提高访问速度的同时，也拥有身份验证以及流量管理等高级功能。基于此，流服务缓存节点采用 Squid 实现代理缓存功能。

11.2.2　Quagga

Quagga 是一个开源路由软件，可以支持多种协议，如 OSPFv2、OSPFv3 以及 BGP 等协议。安装 Quagga 的作用是使得服务器具备路由器的功能，同时支持动态和静态两种路由配置功能。

Quagga 的主要结构是以 Zebra 守护进程作为核心，RIP/OSPF/BGP 等其他动态路由模块作为客户端，如图 11-1 所示。BGP、 OSPF 与 RIP 程序类似，其程序只创建与自身协议相关的 Socket，用户接收与发送协议报文。而添加删除路由的操作由 Zebra 来处理。

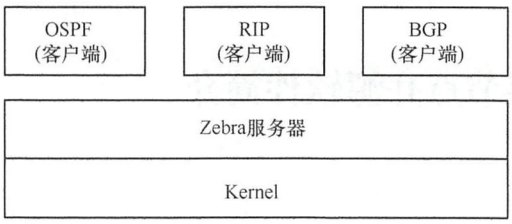


图 11-1　Zebra 示意

Zebra 作为服务器主要提供以下功能：

- 监听内核 Netlink 消息，创建删除接口、IP 地址、路由等信息；
- 设置内核 IP 前传参数；
- 当有内核事件发生时给 OSPF/RIP/BGP 发送通知；
- 为 OSPF/BGP/RIP 等路由程序提供接口，用来进行添加或删除路由的操作；
- 响应客户端的命令，进行配置行为。

而动态路由（OSPF/RIP/BGP）程序作为 Zebra 的客户端，主要提供以下功能：

- 接收 Zebra 通知的事件（接口、路由、IP 地址信息）；
- 自己创建协议的 Socket 收发报文完成协议功能；
- 响应客户端的命令，进行配置行为。

基于 Quagga 的上述特点，采用 Quagga 实现开放式最短路径优先（Open Shortest Path First，OSPF）功能。每个路由器之间可以通过网络接口状态来建立链路状态数据库，并生成最短路径树，然后每个 OSPF 路由器都使用它来构建路由表。

OSPF 可以构建等价多路径，可以将发往目的地址的数据分组通过多路等值的路径进行传输。本书将使用 OSPF 协议来实现多台服务器轮询提供服务。

11.2.3　LVS

Linux 虚拟服务器（Linux Virtual Server，LVS）目前已被广泛使用。现在，Linux 平台标准内核中也包含了 LVS，即在 Linux 系统中可以直接使用 LVS 的各种功能。

目前 LVS 被广泛使用在服务器集群的本地负载均衡中，主要是基于 OSI 模型的第四层（传输层）负载均衡，因此，支持 TCP/UDP 传输协议。LVS 支持多种本地负载均衡策略，并且每种策略都配置简单，使用方便。同时，LVS 也具有稳定可靠的特性，即使集群中某台服务

器无法正常工作，也不会影响到整体的工作效果。LVS 还具备优良的可扩展性，可以把许多低性能的服务器整合起来，对外呈现为一个高性能的服务器。

11.2.4　Keepalived

LVS 通常是结合 Keepalived 一起对外提供服务的。Keepalived 最初是为了 LVS 设计的，专门用于检测集群中每个服务器的状态，根据服务器状态进行集群系统的自动化维护。当集群中某个服务器发生故障或者出现异常情况无法继续提供服务的时候，Keepalived 将会自动将该服务器从服务集群中删除，以此保证集群始终可以对外提供良好的服务。有了 Keepalived，无需人工探测 LVS 集群服务器的负载状态，人工需要完成的只是修复出现故障的服务器。

11.2.5　Nginx

Nginx 是一个轻量级的 Web 服务器 / 反向代理服务器及电子邮件（IMAP/POP3）代理服务器，并在一个 BSD-like 的协议下发行。

Nginx 的功能十分强大，它能够支持大量的并发连接，根据目前的资料显示 Nginx 可以支持高达 50 000 个并发连接数。同时 Nginx 底层都是基于 C 语言进行编写的，因此运行时占用的内存很少，CPU 使用率高。

Nginx 在本书的架构中以代理服务器的角色来接收网站访问连接请求，并处理请求，然后将内容返回给用户。

11.2.6　Lua

Lua 是脚本语言中的一种，其以轻量的特性以及容易耦合其他多种语言的特点，被广泛使用在许多大型游戏的开发中。同时 Lua 本身的语法也比较简单，对于开发人员而言，易于上手，开发效率也极高。

虽然 Lua 有很多的优点，但是它在面对字符串的处理时显得不够强大。因为 Lua 处理字符串时只是调用了 C 库里的接口，没有对 C 库的接口进行补充与扩展。在很多时候，C 库提供的接口功能也十分有限，例如 C 库中就没有对字符串进行切割的接口，都需要开发人员自己去实现。

Lua 也被广泛使用在 Nginx 中，通过在 Nginx 中编写脚本进行逻辑处理、日志分析等。

Nginx 结合 Lua 还具有以下优点。

- 转发次数减少一次：如果使用其他语言来开发服务，肯定会在 Nginx 和服务器中使用一种协议进行通信，这样的话需要再进行一次请求转发。而 Lua 直接运行在 Nginx 中，无需多一次转发便可以直接进行通信。
- 基于事件的响应：由于 Lua 是直接运行在 Nginx 中，Lua 也将继承 Nginx 的所有特性。

11.3 模块设计

本书采用的 CDN 流服务缓存节点如图 11-2 所示。考虑到整个系统的可用性和稳定性，该节点一共由 7 台设备组成，划分为 3 个部分：后端的缓存集群 Squid（3 台设备）、四层的本地负载均衡层 LVS 集群（2 台设备）和七层的本地负载均衡层 Nginx（3 台设备），其中，LVS 集群与 OSPF 路由协议相结合，实现双主模式提供服务。用户的请求到达交换机的时候，通过 OSPF 路由协议将用户请求轮询发送到流服务节点不同的 LVS 服务器上，单台 LVS 根据四层负载均衡的调度策略将请求分流到 LVS 的下一层 Nginx 上，Nginx 获取了用户请求后根据七层负载均衡策略将请求下发到指定的流媒体缓存服务器 Squid 上，并由 Squid 为用户进行服务。整个过程中，通过四层负载均衡和七层负载均衡的配合，达到了减轻后端缓存服务器压力的目的，并可以对服务器进行更好的管理以及实现更好的容灾。

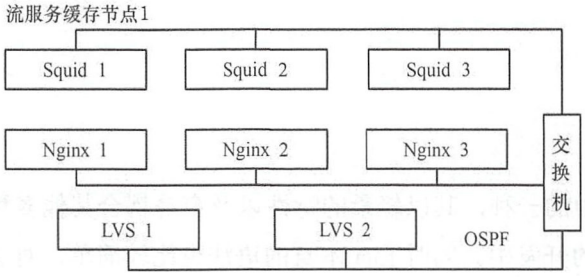


图 11-2　一个 CDN 流服务缓存节点组成

11.3.1 代理缓存（Squid）

Squid 在本系统中的作用是作为缓存，可以缓存互联网中各个网站的数据或将内容提供

商中的内容备份起来，当用户发起访问请求后，可以直接将缓存的内容发送给用户，从而在客户端和服务器端中间起到访问内容加速获取的作用。Squid 在本系统中主要是接收用户的视频请求并做出响应，起到加速视频访问请求的作用，其工作过程如下。

- 当用户的请求到达代理缓存服务器的时候，缓存服务器先解析用户请求，根据解析结果检查缓存服务器中的数据缓存。当缓存中有用户需要的数据内容时，Squid 直接提取数据返回给用户，而无需访问原始网站。
- 当Squid缓存中没有找到用户需要的数据时，Squid会将请求向上一级服务器进行转发，同时 Squid 也将获取上级服务器返回的内容。当 Squid 接收到上级服务器返回的内容时，会先在本地进行备份，然后再将内容发送给用户。当 Squid 再次接收到相同的用户请求时，则直接通过缓存服务器中的备份内容进行响应。

流媒体服务缓存模块的部署模型架构具有以下特性。

（1）架构对称性

架构对称性是指在模块每一层次的设计上，该层的每个节点的功能和作用均等。对于前端的 LVS 集群、中间代理的 Nginx 集群以及后端的 Squid 服务器集群，每一层次中并没有采用类似于 Master 和 Slave 的主从架构。因为在常见的主从结构中，容易导致主节点的压力增加，运营维护的困难也会相对增加。架构的对称性设计不会轻易地因为某个主节点的失效而变得不可用，保证了整个系统的稳定性。

（2）无单点故障

由于该模块架构中采用的对称性设计，每个服务器的地位完全平等。因此，系统的性能不会因为某个设备的失效而导致整个系统不可用。

（3）高性能

在架构中，由于大量的并发和访问量都由前端的本地负载均衡进行决策处理，因此，后端的 Squid 承受的压力比较小。

在本书的流媒体服务器设计中，大部分并发流量都由前端的四层负载均衡和七层负载均衡处理，到达后端的 Squid 并发流量比较小，后端的负载比较稳定。

11.3.2　四层负载均衡（OSPF+LVS+Keepalived）

LVS 在此作为四层负载均衡，它建立在传输层上，主要用于实现客户端流量的分流。LVS

处于流服务缓存节点的最前端交换机。根据 OSPF 协议将用户的请求送达流服务缓存节点后会先传递给 LVS，LVS 根据负载策略将请求分别传递给后端的 Nginx 服务器。LVS 不仅仅可以高效地分流用户请求，同时可以对后台的多台 Nginx 数量起到一个收敛的作用，有利于整个系统的维护与管理。

本书中将用两台 LVS 实现四层负载均衡，所以需要用到 OSPF 路由协议实现 LVS 双主模式的轮询服务，同时也需要用到 Keepalived 来帮助监测 LVS 的健康状况。由于 OSPF 是一个路由器的协议，可以根据配置将流量以轮询的方式分别发给流媒体缓存节点的两台 LVS。通过 Keepalived 与 LVS 相结合，能够实现实时监测 LVS 服务器状态，并且自动将出现异常的 LVS 节点或者故障的 LVS 节点剔除出去，始终保证节点能正常地为用户提供服务。同时在故障节点修复后，无需人工参与，Keepalived 便能将修复后的 LVS 重新添加到节点中。通过 Keepalived 与 LVS 相结合，保证了流媒体服务节点的稳定性与可靠性。

11.3.3 七层负载均衡（Nginx+Lua）

Nginx 在本书的流服务缓存节点中充当七层负载均衡的角色。七层负载均衡也是建立在四层负载均衡之上的，然后再考虑关于应用层的一些需求。对于七层负载均衡而言，除了接收用户请求以外，还可以对用户的 URL 进行进一步的解析，根据解析结果决定负载均衡的策略等。同时 Nginx 可以和 Lua 相结合，通过 Lua 脚本快速解析 URL、日志等，实现不同情况下的本地负载策略。

七层负载均衡的引入可以使整个系统的流量调度更加智能化，也可以使开发人员更高效、便捷地调整调度策略，以更好地满足用户需求。在本书中，在一个节点内，通过应用层的负载均衡，将用户调度到最佳的服务器。

11.4 环境配置

11.4.1 Squid 安装与配置

安装 Squid 的时候需要先关闭防火墙等操作再进行安装，配置如下。

（1）关闭 selinux

```
[root@server~]# vim /etc/sysconfig/selinux
SELINUX=disabled
```

（2）关闭 iptables

```
[root@server~]# /etc/init.d/iptables stop
```

（3）检查 Squid 软件是否安装

```
[root@server~]# rpm -qa|grep squid
```

（4）如果未安装，则使用 yum 方式安装

```
[root@server~]# yum -y install squid
```

（5）设置开机自启动，在 35 级别上自动运行 Squid 服务

```
[root@server~]# chkconfig --level 35 squid on
```

（6）Squid 服务器的配置文件说明

Squid 的主配置文件是 /etc/squid/squid.conf，这里 Squid 配置如下：

```
[root@server~]# vim /etc/squid/squid.conf
http_port 3128      //默认监听的 IP 地址与端口号
cache_mem 64 MB             //额外使用内存量，可根据系统内存在设定，一般为实际内存的1/3，比如内存一共是 200 MB，这里设置 1/3 就是 64 MB
maximum_object_size 4 MB   //设置 Squid 磁盘缓存最大文件，超过 4 MB 的文件不保存到硬盘
minimum_object_size 0 KB    //设置 Squid 磁盘缓存最小文件
maximum_object_size_in_ memory 4096 KB  //设置 Squid 内存缓存最大文件，超过 4 MB 的文件不保存到内存
cache_dir ufs /var/spool/squid 100 16 256   //定义 Squid 的缓存存放路径、缓存目录容量（单位 MB）、一级缓存目录数量、二级缓存目录数量
logformat combined %&gt;a %ui %un [%tl] "%rm %ru HTTP/%rv" %Hs %<st "%{Referer}>h" "%{User-Agent}&gt;h" %Ss:%Sh
access_log /var/log/squid/access.log combined  //log 文件存放路径和日志格式
cache_log /var/log/squid/cache.log                    //设置缓存日志
cache_swap_log /var/log/squid/swap.log squid     //设置交换日志
logfile_rotate 60     //log 轮循 60 天
cache_swap_high 95    //缓存目录使用量大于 95%时，开始清理旧的缓存
cache_swap_low 90     //缓存目录清理到 90%时停止
acl localnet src 192.168.1.0/24    //定义本地网段
```


```
http_access allow localnet          //允许本地网段使用
http_access deny all                //拒绝所有
visible_hostname squid.david.dev    //主机名
cache_mgr wangshibo@huanqiu.com     //管理员邮箱
```

（7）配置 Squid 代理服务器 IP 地址，将 eth1 的 IP 地址修改为 172.19.110.1：

```
[root@server~]# ifconfig eth1 172.19.110.1
```

（8）本书自建 CDN 中的 Squid 主配置文件/etc/squid/squid.conf 如下：183.91.51.65 和 172.19.120.61 分别代表内容库和源站（分别针对视频和页面的回源）。

```
[root@server~]# vim /etc/squid/squid.conf
http_port 3128 transparent
cache_peer 183.91.51.65 parent 80 0 originserver name=a
cache_peer 172.19.120.61 parent 80 0 originserver name=b
cache_peer_domain a video.ctgcdn.net
cache_peer_domain b www.ctgcdn.net
visible_hostname 172.19.110.156
acl manager proto cache_object
acl localhost src 127.0.0.1/32 ::1
acl to_localhost dst 127.0.0.0/8 0.0.0.0/32 ::1
acl localnet src 10.0.0.0/8      //RFC1918 possible internal network
acl localnet src 172.16.0.0/12   //RFC1918 possible internal network
acl localnet src 192.168.0.0/16  //RFC1918 possible internal network
acl SSL_ports port 443
acl Safe_ports port 80 8080      //HTTP
acl Safe_ports port 21           //FTP
acl Safe_ports port 443          //HTTPS
acl CONNECT method CONNECT
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localnet
http_access allow localhost
```


```
http_access allow all
maximum_object_size 2048 MB
cache_dir ufs /data/cache 4096 32 512
cache_swap_log /var/log/squid/swap.log squid
cache_mem 128 MB
hierarchy_stoplist cgi-bin ?
coredump_dir /var/spool/squid
refresh_pattern ^ftp:          1440    20%     10080
refresh_pattern ^gopher:       1440    0%      1440
refresh_pattern -i  (/cgi-bin/|\?) 0     0%      0
refresh_pattern \.(jpg|png|gif|mp3|xml) 1440  50%    2880   ignore-reload
refresh_pattern .       0       20%     4320
```

（9）初始化

```
[root@server~]# squid -z
```

（10）启动 Squid

```
[root@server~]# /etc/init.d/squid start
```

11.4.2　OSPF 安装与配置

（1）需要在流媒体缓存节点的两台 LVS 上都安装 OSPF 软件

```
[root@localhost ~]#yum -y install quagga
```

（2）配置 zerba.conf

```
[root@localhost ~]#cat /etc/quagga/zebra.conf
hostname lvs-route-1
password xxxxxx
enable password xxxxxx
log file /var/log/zebra.log
service password-encryption
```

（3）配置 ospfd.conf

```
[root@localhost ~]#cat  /etc/quagga/ospfd.conf
log file /var/log/ospf.log
log stdout
```


```
log syslog
interface eth0
#ip ospf hello-interval 1                    //本行需注释掉
#ip ospf dead-interval 4                     //本行需注释掉
router ospf
ospf router-id 172.19.110.147
#log-adjacency-changes                       //本行需注释掉
#auto-cost reference-bandwidth 1000          //本行需注释掉
network 172.19.110.149/32 area 0.0.0.100
network 172.19.110.0/24 area 0.0.0.100
```

（4）开启 IP 地址转发

```
[root@localhost ~]#sed -i '/net.ipv4.ip_forward/d' /etc/sysctl.conf
[root@localhost ~]#echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.confsysctl -p
```

（5）开启服务

```
[root@localhost ~]#/etc/init.d/zebra start
[root@localhost ~]#/etc/init.d/ospfd start
[root@localhost ~]#chkconfig zebra on
[root@localhost ~]#chkconfig ospfd on
```

11.4.3 LVS 安装

以操作系统 CentOS 为例，LVS 的安装源存储在 Linux 的配置文件中，可以通过 CentOS 的 yum 命令直接获取 LVS 的安装包。

为四层负载均衡上每一台设备安装 LVS，顺序如下。

（1）依赖包安装

执行以下命令检查是否安装对应的依赖包：

```
[root@localhost ~]#rpm -qa|grep popt
```

如果没有，则需要依次安装，具体如下：

```
[root@localhost ~]#yum install popt
[root@localhost ~]#yum install popt-devel
[root@localhost ~]#yum install popt-static
[root@localhost ~]#RPM-qa|grep libnl
```


如果没有上述 libnl 包，则需要依次安装，具体如下：

```
[root@localhost ~]#yum install libnl
[root@localhost ~]#yum install libnl-devel
```

（2）ipvsadm 安装

```
[root@localhost ~]#wget http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.26.tar.gz
[root@localhost ~]#tar zxvf ipvsadm-1.26.tar.gz
[root@localhost ~]#ln -s /usr/src/kernels/2.6.32-71.el6.x86_64 /usr/src/linux        //注意：对于每个系统该路径可能会不一样
[root@localhost ~]#cd ipvsadm-1.26
[root@localhost ~]#make
[root@localhost ~]#make install
```

LVS 安装成功。

11.4.4　Keepalived 安装与配置

（1）四层负载均衡层上，每一台机器都需要安装 Keepalived，顺序如下：

```
[root@localhost ~]#wget http://www.keepalived.org/software/keepalived-1.2.7.tar.gz
[root@localhost ~]#tar zxvf keepalived-1.2.7.tar.gz
[root@localhost ~]#cd keepalived-1.2.7
[root@localhost ~]#./configure
[root@localhost ~]#make
[root@localhost ~]#make install
```

Keepalived 安装成功。

在安装完后，需要对 Keepalived 的配置文件 keepalived.conf 进行修改，将需要添加的 VIP 以及 real server 添加上。

（2）备份并打开配置文件，修改部分内容具体如下：

```
[root@localhost ~]#cp /etc/keepalived/keepalived.conf /etc/keepalived/keepalived.conf.bak
[root@localhost ~]#vim /etc/keepalived/keepalived.conf
```


（3）VRRP 实例定义部分

```
vrrp_instance VI_1 {
    state MASTER  //Keepalived 的角色，MASTER 表示主机是主服务器，BACKUP 表示备用服务器
    interface eth0          //指定监测的网络网卡
    virtual_router_id 51    //虚拟路由标识
    priority 100            //定义优先级，数字越大，优先级越高，MASTER 的优先级必须大于 BACKUP 的优先级
    advert_int 1            //设定主备之间检查时间，单位为秒（s）
    authentication {        //设定验证类型和密码
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {     //可以设置多个虚拟 IP 地址，每行一个
        170.19.110.149
    }
}
```

（4）虚拟服务器部分

```
virtual_server 170.19.110.149 80 {
    delay_loop 6            //设定运行情况检查时间，单位为秒（s）
    lb_algo rr              //负载均衡算法，rr 即轮询算法
    lb_kind DR              //设置 LVS 负载机制，NAT、TUN、DR 共 3 种模式可选
    nat_mask 255.255.255.0
    persistence_timeout 0   //会话保持时间
//有了这个会话保持功能，用户的请求会被一直分发到某个服务节点
//如果用户在动态页面 50 s 内没有任何动作，那么后面就会被分发到其他节点
//如果用户一直有动作，不受 50 s 限制
    protocol TCP //协议
```

（5）real server 部分

```
  real_server 170.19.110.152 80 {
      weight 1     //服务节点权值，数字越大，权值越高
    //权值的大小可以为不同性能的服务器分配不同的负载
```


```
    //这样才能有效合理地利用服务器资源
        TCP_CHECK {       //状态检查部分
          connect_timeout 3      //无响应超时
          nb_get_retry 3         //重试次数
          delay_before_retry 3    //重试间隔
          connect_port 80         //连接端口
        }
}
```

（6）启动 Keepalived

```
[root@localhost ~]#/etc/init.d/keepalived start
[root@localhost ~]#chkconfig keepalived on
```

11.4.5　Nginx 安装与配置

（1）安装 Nginx 所需要的依赖库

```
[root@localhost ~]# yum install -y pcre pcre-devel
[root@localhost ~]# yum install -y zlib zlib-devel
[root@localhost ~]# yum install -y openssl openssl-devel
```

（2）去 Nginx 的官网下载对应版本的 Nginx，官网地址为 http://nginx.org

本文中采用的 Nginx 是 1.11.1 版本。对下载的 Nginx 进行解压以及编译安装，具体如下：

```
[root@localhost ~]# tar -zxvf  nginx-1.11.1.tar.gz
[root@localhost ~]# cd nginx-1.11.1
[root@localhost ~]# ./configure
[root@localhost ~]# make
[root@localhost ~]# make install
```

Nginx 默认安装位置是”/usr/local/nginx”。

（3）防火墙开放 Nginx 的默认端口 80，具体如下：

```
[root@localhost ~]# /sbin/iptables -I INPUT -p tcp --dport 80 -j ACCEPT
[root@localhost ~]# /etc/init.d/iptables save
[root@localhost ~]# service iptables restart
```

（4）在浏览器中访问本机 IP 地址，若出现标题为“Welcome to nginx!”的页面，则证明

安装成功。

11.4.6 lua-nginx-module 模块安装

lua-nginx-module 是一个 Nginx HTTP 模块，它把 Lua 解析器内嵌到 Nginx，用来解析并执行 Lua 语言编写的网页后台脚本，下面将介绍如何在 Nginx 中安装 lua-nginx-module 模块。

（1）下载安装 LuaJIT 2.1（2.0 或者 2.1 都是支持的，官方推荐 2.1）

对 LuaJIT 安装包进行解压以及安装：

```
[root@localhost ~]#tar -zxf LuaJIT-2.1.0-beta3.tar.gz
[root@localhost ~]#cd LuaJIT-2.1.0-beta3
[root@localhost ~]#make prefix=/usr/local/luajit
[root@localhost ~]#make install prefix=/usr/local/luajit
```

（2）下载 ngx_devel_kit（NDK）模块，不需要安装

```
[root@localhost ~]#cd /usr/local/src
[root@localhost  ~]#wget  https://github.com/simpl/ngx_devel_kit/archive/
v0.2.19.tar.gz
[root@localhost ~]#tar -xzvf v0.2.19.tar.gz
```

（3）下载最新的 lua-nginx-module 模块，不需要安装

```
[root@localhost ~]#cd /usr/local/src
[root@localhost    ~]#wget    https://github.com/openresty/lua-nginx-module/
archive/v0.10.2.tar.gz
[root@localhost ~]#tar -xzvf v0.10.2.tar.gz
```

（4）进入 Nginx 的安装目录，设置环境变量

```
[root@localhost ~]#export LUAJIT_LIB=/usr/local/luajit/lib
[root@localhost ~]#export LUAJIT_INC=/usr/local/luajit/include/luajit-2.1
```

（5）对 Nginx 重新进行编译安装

```
[root@localhost~]#./configure--with-ld-opt="-Wl,-rpath,/usr/local/luajit/lib"\
--add-module=/home/testcdn/soft/ngx_devel_kit-0.3.0 \
--add-module=/home/testcdn/soft/lua-nginx-module-0.10.8
[root@localhost ~]#make -j2
[root@localhost ~]#make install
```


（6）查看是否编译成功

在“/usr/local/nginx/conf/nginx.conf”中加入如下代码：

```
location /hello_lua {
     default_type 'text/plain';
     content_by_lua 'ngx.say("hello, lua") ';
}
```

（7）重启 Nginx

```
[root@localhost ~]#service nginx restart
```

访问 172.19.110.152/hello_lua 会出现“hello, lua”，则表示安装成功。

本书中将结合 Nginx 和 Lua 进行视频文件 URL 请求的重定向，由 Lua 脚本进行 URL 解析，将符合条件的请求重定向到后端的缓存服务器上。

（8）修改 Nginx 的配置文件

本书中位于“/usr/local/nginx/nginx.conf”中，新增用于重定向的 Lua 脚本：

```
server{
       listen  5000;
       server_name cdn.cachenow.net;
       index index.html index.htm index.php;
       root /home/gsta;
       location ~ / {
              default_type 'text/plain';
              rewrite_by_lua_file /usr/local/nginx/conf/vhost/test-1.lua;
     }
}
```

从配置文件中可以看出，Nginx 监听的是 5000 端口，当请求到达 Nginx 的 5000 端口时，将根据 test-1.lua 脚本进行请求的重定向。

（9）具体的重定向规则可见 test-1.lua，如下：

```
local key = ngx.re.sub(ngx.var.request_uri, "^/(.*)/(.*)", "$1", "o")
local req=ngx.re.sub(key, "^(.*)", "http://172.19.110.156:3128/reload/$1", "o")
ngx.redirect(req)
```

可以根据需求配置不同的重定向规则，这里的例子是将获取 URL 请求都重定向到 172.19.110.156.0 的 3128 端口，172.19.110.156 是 Squid 缓存服务器的 IP 地址。

第 12 章

内容库模块的设计

内容存储设备作为整个自建 CDN 中的重要部分，通过将源站需要加速的内容放在 CDN 中进行分布式多副本存储，从而实现系统存储资源、计算资源、带宽资源的合理利用，这就是内容库。通过与源站对接，内容库主要提供包括内容注入、内容存储、内容管理、内容分发以及内容定位等功能。本章将重点介绍 CDN 内容库模块的搭建。

12.1 内容库的特性

内容库通过内容注入指令将源站内容注入内容库设备中，内容库对注入的内容建立多副本，并管理内容，配置内容分发策略，将多个副本内容分发到不同的节点中。

12.2 内容库开源软件简介

12.2.1 FTP

文件传输协议服务器（File Transfer Protocol Server，FTP 服务器），是指利用 FTP 对文件进行一系列上传和下载操作的服务器。

FTP 服务器是基于 C/S 模式的系统，即存在客户端与服务器。用户可以从支持 FTP 的客户端上发出连接请求到远端的 FTP 服务器上。FTP 服务器获取客户端的请求后，可以执行用户发出的命令，并将结果返回给客户端。通常使用 FTP 服务器进行上传和下载两种文件操作方式。上传文件就是将文件从自己的计算机中复制到远端的服务器中；下载文件就是从远端服务器中复制文件到本地计算机中。

12.2.2　Ceph

目前，分布式存储软件有很多，Ceph 作为其中最热门的存储软件，提出了针对块存储、文件存储、对象存储的对应解决方案。Ceph 的功能很强大，它可以提供一个高可扩展性的、高效读写的且没有单点故障的分布式文件存储系统。

Ceph 有集群可靠性、集群扩展性、数据安全性、接口统一性这 4 个特性，具体如下。

- 集群可靠性：Ceph 的架构设计可以保证数据写入的过程稳定可靠，即使是在出现意外的时候，数据也不会丢失。
- 集群可扩展性：Ceph 的集群可扩展性包括集群规模和存储容量这两个方面的扩展。
- 数据安全性：Ceph 可以保证集群中存储的数据在遇到极端情况下（比如，宕机或者停电）也不会丢失，可以做到自动恢复数据。
- 接口统一性：Ceph 针对 3 种存储（块存储、对象存储以及文件存储）都提出了解决方案，所以目前支持市面上所有流行的存储类型，覆盖面广泛。

如图 12-1 所示，Ceph 系统自上而下可以分为 4 个层次，具体介绍如下。

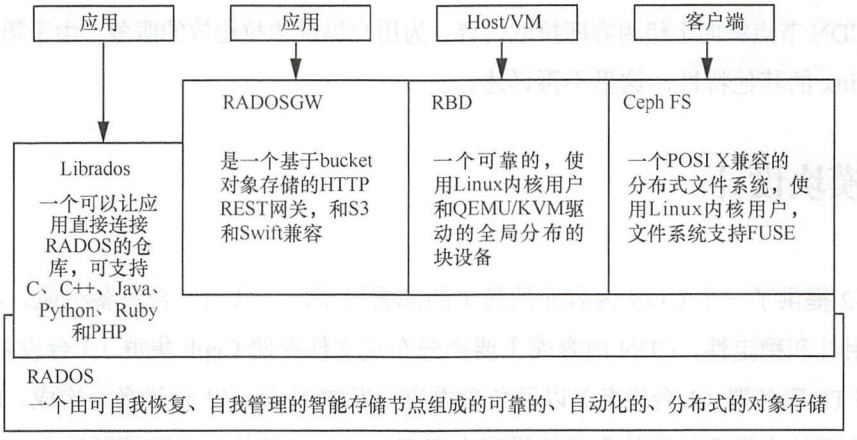


图 12-1　Ceph 总体架构

（1）基础存储系统 RADOS

基础存储系统 RADOS（Reliable Autonomic Distributed Object Store）是一种可靠的、自动化的、分布式的对象存储，为了存储大量的数据，需要保证有大量的存储空间，所以，RADOS 由许多的存储节点构成。RADOS 的存在也保证了 Ceph 的高可扩展性、高可靠性以及高自动化性。由此可看出，RADOS 是 Ceph 的基础部分。

（2）基础库 Librados

Librados 的存在可以使得开发人员直接基于 RAODS 开发，可以直接调用 API 来控制对象的存储。使用 Librados 开发的机器可以使用本地的 Librados 进行开发，开发人员在调用了本地的 Librados API 之后再通过 Socket 通信方式与 RADOS 集群中其他节点进行通信。

（3）高层应用接口

RADOS GW（RADOS Gateway）、RBD（Reliable Block Device）和 Ceph FS（Ceph File System）构成了这一层。

其中，RADOS GW 是一个提供与 Amazon S3 和 Swift 兼容的 RESTful API 的网关，应用开发可以使用此部分。Ceph FS 是一个 POSIX 兼容的分布式文件系统。目前 Ceph FS 还处于一个发展阶段，功能还没有稳定下来，所以生产环境中大多不使用。

（4）应用层

这一层提供的功能可以保证对不同需求的开发按需使用 Ceph 的接口进行开发工作。

此外，Nginx 在内容库模块里起到了内容分发的作用，当下级 CDN 节点发生未命中请求时，下级 CDN 节点将向上级内容库拉取内容，为用户提供边拉边放的服务。由于第 11.2.5 节已介绍过 Nginx 的其他特性，这里不再详述。

12.3 模块设计

图 12-2 提供了一个 CDN 内容库的简单的部署模型，一共由 5 台设备组成。考虑到整个系统的可用性和稳定性，CDN 内容库主要由分布式文件存储 Ceph 集群（3 台设备）、负责内容注入的 FTP 服务器（1 台设备）以及负责内容分发的 Nginx（1 台设备）构成。FTP 客户端可以用来将源站中需要加速的资源传送到内容库。例如，图片、视频等源站内容可以传送到

分布式存储系统 Ceph 中。Ceph 系统接收到源站的文件后，进行多副本复制，并下发到多个存储设备中，后续内容库的读写将由 Ceph 负责。并在流服务缓存节点内容未命中时，利用 Nginx 实现 HTTP 下载内容，为用户提供边拉边放的服务。

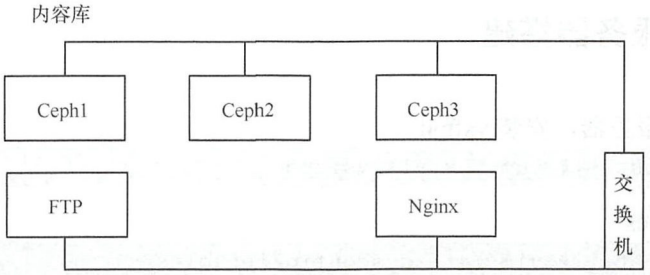


图 12-2　CDN 内容库部署模型

12.3.1　内容注入（FTP）

FTP 用于互联网上控制文件的双向传输。本书中采用 FTP 的方式，将源站内容注入内容库，传输速度快，文件能够准确到达。

12.3.2　分布式内容存储（Ceph）

选取 Ceph 作为内容库的文件存储系统。Ceph 是一个具有高性能、高可靠性以及可扩展性的存储系统，可以支持 3 种方式的存储：对象存储、块存储、文件存储。同时由于 Ceph 系统中的文件是多副本的，因此，在读写操作的时候能够做到高度并行化。同时，多副本能够跨主机、机架、机房、数据中心存储，也保证了数据的安全可靠和强容错性，做到存储节点自动管理与修复。

12.3.3　内容分发（Nginx）

Nginx 在内容库模块里充当 HTTP 文件下载服务，当下级 CDN 节点发生未命中请求时，下级 CDN 节点将向上级内容库发起下载请求，内容库将内容分发到下级节点。

12.4 环境配置

12.4.1 FTP 服务器搭建

（1）选择一台服务器，安装 vsftpd

```
[root@localhost ~]# yum install vsftpd
```

（2）编辑 iptables

```
[root@localhost ~]# vim /etc/sysconfig/iptables
-A INPUT -m state --state NEW -m tcp -p tcp --dport 21
[root@localhost ~]# service iptables restart
```

（3）配置 vsftpd

```
[root@localhost ~]# vim  /etc/vsftpd/vsftpd.conf
anonymous_enable=NO      //设定不允许匿名用户访问
```

（4）添加 FTP 用户

```
[root@localhost ~]# vim /etc/vsftpd/vsftpd.conf
chroot_list_enable=YES
chroot_list_file=/etc/vsftpd/chroot_list
```

（5）添加 FTP 用户 home 目录

```
[root@localhost ~]#useradd -d /var/www/html -g ftp -s /sbin/nologin test
```

（6）设置用户密码

```
[root@localhost ~]#passwd test
```

（7）修改权限

```
[root@localhost ~]#chmod 777 /var/www/html -R
[root@localhost ~]#chown test:ftp /var/www/html
```

（8）启动 vsftpd

```
[root@localhost ~]#service vsftpd restart
```

最后，使用 test 用户即可登录。

12.4.2　Ceph 安装与配置

（1）关闭 selinux

```
[root@localhost ~]# vim /etc/selinux/config
```

修改 SELINUX=disabled：

```
[root@localhost ~]# vim /etc/fstab
```

注释下一行：

```
#UUID=xxxxxxx /home
```

重启电脑。

（2）配置 Ceph yum 源

```
[root@localhost ~]# vim /etc/yum.repos.d/ceph.repo
[ceph-noarch]
name=Ceph noarch packages
baseurl=http://ceph.com/rpm-{ceph-stable-release}/{distro}/noarch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

（3）更新源并且安装 ceph-deploy

```
[root@localhost ~]# yum update && yum install ceph-deploy -y
```

（4）配置各个节点的 hosts 文件

```
[root@localhost ~]# cat /etc/hosts
127.0.0.1  localhost localhost.localdomain localhost4 localhost4.localdomain4
::1        localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.7.2   sh_ceph1
192.168.7.3   sh_ceph2
192.168.7.4   sh_ceph3
192.168.7.5   sh_ceph4
```

（5）配置各节点 SSH 无密码登录，输入 ssh-keygen 命令，然后命令行会输出以下内容：

```
[root@localhost ~]# ssh-keygen
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
```

（6）复制 key 到各个节点

```
[root@localhost ~]# ssh-copy-id sh_ceph1
[root@localhost ~]# ssh-copy-id sh_ceph2
[root@localhost ~]# ssh-copy-id sh_ceph3
[root@localhost ~]# ssh-copy-id sh_ceph4
```

（7）在执行 ceph-deploy 的过程中，可以将产生的配置文件放入新建的目录中

```
[root@localhost ~]# mkdir ceph-conf
```

（8）创建集群，部署新的 monitor 节点

```
[root@localhost ~]# ceph-deploy new {initial-monitor-node(s)}
```

例如：

```
[root@localhost ~]# ceph-deploy new sh_ceph1
```

执行成功后目录下会生成如下 3 个文件：

```
ceph.mon.keyring , ceph.conf, ceph.log,可以根据需要修改 ceph.conf
```

（9）安装 Ceph 到各个节点

```
[root@localhost ~]# ceph-deploy install client sh_ceph1 sh_ceph2 sh_ceph3
```

（10）激活监控节点

```
[root@localhost ~]#ceph-deploy mon create-initial
```

执行成功后本地生成 3 个 key 文件：

```
ceph.bootstrap-mds.keyring, ceph.bootstrap-osd.keyring, ceph.client.admin.
keyring
```

（11）初始化磁盘

例如：

```
[root@localhost ~]# ceph-deploy disk zap sh_ceph1:sdb
```

（12）准备 OSD

例如：

```
[root@localhost ~]# ceph-deploy osd prepare sh_ceph1:sdb1:sdc
```

（13）激活 OSD

例如：

```
[root@localhost ~]# ceph-deploy osd activate sh_ceph1:sdb1:sdc
```

（14）配置 mds 服务

```
[root@localhost ~]# ceph-deploy mds create sh_ceph1
```

（15）创建一个 ceph 文件系统

```
[root@sh_ceph1 ~]# ceph osd pool create cephfs_data 10
[root@sh_ceph1 ~]# ceph osd pool create cephfs_metadata 10
[root@sh_ceph1 ~]# ceph fs new cephfs cephfs_metadata cephfs_data
[root@sh_ceph1 ~]# ceph mds stat
```

（16）同步配置文件

```
[root@localhost ~]# ceph-deploy admin node1 node2 node3
```

（17）查看集群健康状态

```
[root@sh_ceph1 ~] ceph health
```

12.4.3　Nginx 安装与配置

详见第 11.4.4 节，这里不再详述 Nginx 的安装与配置方法。

第 13 章

全局用户请求调度模块的设计

CDN 作为一个内容分发网络，需要制定分发规则才能够为用户提供更好的服务。这里需要一个请求调度模块来为用户建立和管理调度规则，即用户请求调度模块（RR）/全局负载均衡（GSLB），主要为了完成用户的访问调度，为用户分配合适的流媒体节点。

首先，将介绍基于 DNS 层面的调度，最终结果是将域名的解析权完全交给 RR/GSLB，接下来，将介绍基于 HTTP 层面的调度。此时，RR 会根据用户的 HTTP 请求找出可用的流服务节点，并将 HTTP 请求转发给该节点。

13.1 基于 DNS 的流量管理服务设计

RR 中基于 DNS 层面的用户请求调度模块在本书中设计为流量管理模块，是一种基于智能 DNS 的全球负载均衡服务，实现的功能与第 8 章的多租户 TMS 相同，实现方式是通过在 CDN 间进行快速可靠的流量分布调度，可以独立部署，也可以与第三方 CDN 进行协作服务。下面具体介绍该流量管理服务需要包含的部件。

13.1.1 基于 DNS 流量管理的开源软件简介

流量管理系统，主要是通过 BIND（Berkeley Internet Name Domain）结合 MySQL 数据

库来实现。

（1）BIND

BIND 是一款开放源代码的 DNS 服务器软件，目前，很多 DNS 服务器都采用了 BIND。

在流量管理模块设计中，BIND 能够实现 DNS 协议，将域名和 IP 地址互相映射，可以提供域名的查询功能。

（2）MySQL 数据库

MySQL 是目前使用十分广泛的数据库应用软件。它可以运行在多个平台上，如 UNIX、Linux、Windows。它具有 C/S 模式（即客户端和服务器的模式结构）的分布式数据库管理系统。MySQL 以其使用简单、快捷、可靠性好等受到了各大公司的青睐。MySQL 也可以被不同语言所调用，如 C、C++、Java 等。MySQL 在存储机制的设计上，也提供了事务和非事务的存储机制；在核心线程上，MySQL 支持多线程、多 CPU，在流量管理模块设计中，MySQL 用于存储数据，供 DNS 服务调用。

13.1.2　模块设计

基于 DNS 的流量管理模块主要包括 DNS 服务与调度策略配置，其中，涉及的功能说明如下。

- BIND 支持标准 DNS 协议，根据协议接受用户的请求，并返回给用户查询结果。智能 DNS 的实现通常采用 BIND+DLX+MySQL 的组合模式。
- MySQL 可以与 BIND 相结合，通过 MySQL 来配置不同的调度策略。为了方便配置，通常还设置流量管理配置页面，其主要用来修改配置（包括 DNS IP 地址配置）、查看用户访问记录、管理节点以及账号等。

13.1.3　环境配置

EDNS，即智能 DNS，是 Google 提交的 DNS 扩展协议，允许 DNS 查询请求中除了本地 DNS IP 地址，还包括用户自身的 IP 地址，因此 DNS 可以获取用户真实的 IP 地址进行调度。目前支持 EDNS 协议的有 Google、OpenDNS 等。Google 的 DNS 能自动检测是否智能，并主动向流量管理系统发送 EDNS 请求；OpenDNS 则需要提出申请，将 DNS 加入它的白名单，才能向流量管理系统发送 EDNS 请求。

实现方案用的是 BIND-9.8.3，通过打补丁的方式，加上支持 GeoIP（地理信息数据库）

和 EDNS 的补丁代码，然后编译和安装。

（1）增加 GeoIP 补丁

BIND 需要根据 DNS 请求的地理位置信息返回查询结果，增加支持 GeoIP 的 patch 后，BIND 就可以直接查询 GeoIP 数据库获取用户的地理位置，无须手动收集和配置 IP 地址信息。

（2）增加 EDNS 补丁

如果用户用了公共的 DNS，例如 Google 或者 OpenDNS，那么可能用户的真实 IP 地址在中国，但是由于 DNS IP 地址在美国（Google），所以用户就被定位成美国用户（实际上不是离用户最近的服务器）。因此，BIND 需要增加支持 EDNS 的 patch，这样 DNS 请求中会增加用户真实 IP 地址，BIND 根据用户真实 IP 地址而不是 DNS IP 地址来调度。

（3）增加支持 dig 的补丁

BIND 还需要增加支持 dig 的 patch，以模拟 EDNS 发出的 DNS 查询请求。

具体的 BIND-9.8.3+EDNS+GeoIP 补丁的安装步骤如下。

（1）下载 BIND-9.8.3 的源代码

```
[root@localhost~]#
http://download.chinaunix.net/download.php?id=40370&ResourceID=6
```

解压：

```
tar zxvf bind-9.8.3
```

BIND-9.8.3 需要增加 GeoIP 的 patch，前提是要有 GeoIP C API，还要用到 GeoIP 数据库。

（2）下载 MaxMind 的 GeoIP API C（版本为 GeoIP 1.6.0）

```
[root@localhost~]#https://github.com/maxmind/geoip-api-c/releases/downloa
d/v1.6.0/GeoIP-1.6.0.tar.gz
```

解压 GeoIP-1.6.0：

```
tar zxvf GeoIP-1.6.0.tar.gz
```

（3）安装 GeoIP-1.6.0

```
[root@localhost ~]#./configure
[root@localhost ~]#make
[root@localhost ~]#make install
```

注：GeoIP-1.6.0 安装在/usr/local/lib，包含 libGeoIP.so、libGeoIP.so.1 等库，供 BIND 调用。

（4）下载 GeoIP 的数据库

GeoIP 数据库包括 Geo IP Country 和 Geo IP City（国家和城市的 IP 地址数据库）。

安装 GeoIP 的数据库（具体参见 http://dev.maxmind.com/geoip/legacy/install/country/）。

（5）下载 Country 数据库和 City 数据库（免费）

```
[root@localhost ~]#wget -N http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz
wget -N http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
```

解压数据库：

```
[root@localhost ~]# gunzip GeoIP.dat.gz
[root@localhost ~]#gunzip GeoLiteCity.dat.gz
```

把数据库放到默认安装目录：

```
[root@localhost ~]#mv GeoIP.dat /usr/local/share/GeoIP/
[root@localhost ~]#mv GeoLiteCity.dat.gz /usr/local/share/GeoIP/
```

把 City 数据库改名（否则 BIND 找不到 City 数据库）：

```
[root@localhost ~]#mv /usr/local/share/GeoIP/GeoLiteCity.dat  /usr/local/share/GeoIP/GeoIPCity.dat
```

（6）下载 BIND-9.8.3-geoip patch

```
[root@localhost~]#https://bind-geoip.googlecode.com/files/bind-9.8.3-geoip-1.3.patch
```

将 BIND-9.8.3-geoip-1.3.patch 放到 BIND-9.8.3 目录下（目前目录为：/usr/local/src/bind-9.8.3）。

进入 BIND-9.8.3 目录，打补丁，注意查看打补丁是否全部成功：

```
[root@localhost ~]#patch -p0 -b < bind-9.8.3-geoip-1.3.patch
```

自动配置 GeoIP 的 patch：

```
[root@localhost ~]#autoconf
```

（7）下载 0001-EDNS0-client-subnet-support.patch（这里 EDNS patch 支持的 BIND 版本为 9.8.x）

```
[root@localhost ~]#https://gist.github.com/vincentbernat/6524506
```

将 0001-EDNS0-client-subnet-support.patch 放到 BIND-9.8.3 目录下（目前目录为：/usr/local/src/bind-9.8.3）。

进入 BIND-9.8.3 目录，打补丁，注意查看打补丁是否全部成功，如果有失败，需要根据失败信息，核对具体代码文件，手工进行修改：

```
[root@localhost ~]#patch -p1 -b < 0001-EDNS0-client-subnet-support.patch
```

（8）下载 dig patch（版本为 9.9.3）

```
[root@localhost ~]#http://wilmer.gaa.st/edns-client-subnet/
```

将 dig patch 放到 BIND-9.8.3 目录下（目前目录为：/usr/local/src/bind-9.8.3）。

进入 BIND-9.8.3 目录，打补丁，注意查看打补丁是否全部成功，如果有失败，需要根据失败信息，核对具体代码文件，手工进行修改：

```
[root@localhost ~]#patch -p0 -b < dig.patch
```

（9）编译 BIND-9.8.3

```
[root@localhost ~]#./configure
prefix=/edns/bind      --disable-ipv6      --disable-openssl-version-check
--with-dlz-mysql --enable-largefile --enable-threads=no --with-geoip=/usr/local
--with-geoip-debug
[root@localhost ~]#make
[root@localhost ~]#make install
```

13.2 基于 HTTP 的应用层调度服务设计

上文介绍了基于DNS层面的调度过程，最终结果是将DNS的解析权完全交给RR/GSLB，接下来介绍基于 HTTP 层面的调度。

当解析权交给 RR 后，RR 会根据用户的 HTTP 请求找出可用的流服务节点，并将 HTTP 请求转发给该节点。

13.2.1 基于 HTTP 调度的开源软件简介

HTTP 层面的请求调度是基于 HTTP 进行的，调度模块将获取到的 HTTP 请求，转发到真正可用的流服务节点。这里的 HTTP 调度模块是采用 Nginx 实现的。关于 Nginx 的其他特性，本书在前面已经介绍过，这里不再赘述。

13.2.2 模块设计

Nginx 调度模块接收用户的 HTTP 请求后，通过 CDN 节点管理表，将请求转发到高性能

的可用节点，即转发 HTTP 请求给相应的流服务缓存节点，并由该节点直接为用户提供服务；若流服务缓存节点未命中，而仍将该节点的 VIP 分配给用户，则该流服务缓存节点需要向上级节点拉取内容，并为用户提供边拉边放的服务。此外，该模块能够直接灵活地配置 Nginx 的调度策略，本节设置的调度策略轮询各个流服务缓存节点。

13.2.3　环境配置

本节介绍在 Nginx 上配置基于 HTTP 的服务，实现各个服务节点间的调度功能。在 Nginx 上配置 upstream（上游），转发 HTTP 请求给相应的流服务缓存节点，此处 172.19.110.148、172.19.110.149 分别为不同流媒体缓存节点的 LVS 集群的 VIP，通过配置请求到达调度模块 RR/GSLB，这里默认设置轮询方式为将请求转发到其中一个流媒体服务节点。如需增加节点可在 backend 中添加 server，具体如下：

```
upstream backend {
  server 172.19.110.148:80;
  server 172.19.110.149:80;
     }

     server {
        listen       80;
        server_name  localhost;

        location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_buffering off;
            proxy_pass http://backend ;
}
```

第 14 章

网络管理模块的设计

现今互联网行业发展迅速，底层支撑互联网服务的服务器等各种硬件设备规模越来越庞大，如何有效地管理这些设备、实时发现其潜在的隐患、获取其运行的状态，就显得至关重要。CDN 的网络管理模块以用户体验为中心，进行网络监控、故障查找以及提高业务稳定性等工作，从而提高 CDN 的整体服务质量。本节主要介绍网络管理模块的主要工作过程和基本部件。

14.1 网络管理工作流程

网络管理模块在整个 CDN 中从服务器监控开始，进行日志分析以及故障统一告警的一系列工作流程。

（1）服务器监控

服务器监控部分主要监控自建 CDN 的各种节点信息，可以监控节点的各种资源：包括服务器负载、磁盘使用率、服务状态等。同时，还可以通过一些处理程序预防节点的故障发生。

（2）日志分析与处理

服务器监控后，其目的在于排除故障，那么接下来需要进行日志数据分析。日志可以大

体划分为系统日志和网站日志两类。系统日志中记录关于服务器系统的各种指标信息；而网站日志则记录服务器接收处理请求以及运行时错误等各种原始信息文件。

CDN 的网络管理模块通过对日志进行统计、分析、综合，就能有效地掌握 CDN 全网运行状况，发现和排除错误原因，了解客户访问分布等，更好地加强系统的维护和管理。同时，日志也是了解 CDN 资源访问频率的最佳途径。通过这个文件，可以了解用户在特定时间请求的资源以及获取的内容等。

在分析日志时，需要分析的内容包括访问次数、停留时间、抓取量、目录抓取统计、页面抓取统计、HTTP 状态码等。

（3）告警平台展示

在分析日志数据后，成功定位故障问题，这时需要一个平台进行告警展示、通知用户，因此，统一告警平台也是必不可少的。

14.2　网络管理开源软件简介

14.2.1　Zabbix

Zabbix 软件是一个提供图形化界面的分布式系统监控开源软件。Zabbix 的底层运行主要基于 C/S 模式，通过 C/S 模式收集监控数据，然后通过 B/S 模式展示监测数据。Zabbix 可以监控各个主机的性能指标，还可以监测应用（例如数据库的性能以及 FTP 等通用协议）。Zabbix 采用服务器和客户端的方式，可以通过服务器端统一收集展示各个分布式节点的监控数据。在使用 Zabbix 的过程中，还可以调用 Zabbix 服务器端的 API 来编写插件，可以自定义监控项以及报警级别。

与其他的监控软件相比，Zabbix 具有以下几点优势：

- 后台运行采用客户端和服务器的结构，服务器端只进行数据采集和展示作用，对设备的性能要求低；
- Zabbix 可以支持的设备多样，支持多种监控模块；
- Zabbix 采用的是分布式架构，可以自动发现功能，实现自动化的监控；
- 可以按照需求调用 Zabbix 的接口进行插件的编写，可扩展性强；

- 当需要监控的项目比较多时，被监控客户端会调整策略，主动向服务器端获取监控的项目，然后再将数据上传到服务器端，对服务器端的负载比较小；
- 可与其他系统相互结合，使用方便。

由于 Zabbix 采用的是客户端与服务器的结构，所以需要在被监控的主机上都安装客户端，并且收集到的数据都存放在数据库里，将会在数据库上存在瓶颈。

Zabbix 的架构如图 14-1 所示。可以看出，Zabbix 由以下几个组件构成。

- Zabbix 服务器（Zabbix Server）：用于与被监控机器进行建立连接，并接收客户端发送的监控数据，服务器端负责管理数据的收集与分析以及操作数据进行一系列的展示；
- 数据库存储（Database Storage）：用于存储所有被监控客户端上传的数据信息，用于存储所有的配置信息；
- Web 界面（Web Interface）：Zabbix 的图形化接口，通常在安装了 Zabbix 服务器的机器上运行；
- 代理服务器（Proxy）：这个组件是可以根据需要进行选择的，作为代理收集部分需要收集的数据，并统一发送给 Zabbix 服务器端；
- 客户端（Agent）：安装在需要被监控的机器主机上，用于收集本地的性能监控数据，然后将数据发送给服务器端或者代理端。

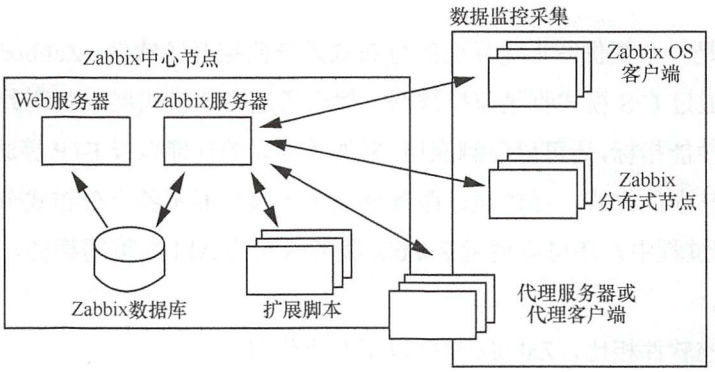


图 14-1　Zabbix 架构

Zabbix 的大致工作过程是：Zabbix 的客户端被安装在需要监控的主机上，然后客户端模块定期收取主机上的各个监控数据，将收集的数据发送给 Zabbix 的服务器端，服务器端获取到数据后存入数据库中，然后 Zabbix Web 将会读取数据在前端进行展示。关于客户端收集监

控数据的方式有两种：主动和被动。不同方式的详细过程如下。

在主动模式下，Zabbix 的客户端将主动请求服务器端获取监控项的列表，并自动将需要检测的数据提交给服务器/代理端；但是在被动的情况下，服务器会主动向客户端发送请求获取监控的数据。

其中，主动监测通信过程如下：

（1）获取 active items 列表；

（2）客户端打开 TCP 连接（主动检测变成客户端打开）；

（3）客户端请求 items 检测列表；

（4）服务器端返回 items 列表；

（5）客户端处理响应；

（6）关闭 TCP 连接；

（7）客户端开始收集数据；

（8）主动检测提交数据；

（9）客户端建立 TCP 连接；

（10）客户端提交 items 列表收集的数据；

（11）服务器端处理数据，并返回响应状态；

（12）关闭 TCP 连接。

被动监测通信过程如下：

（1）服务器端打开一个 TCP 连接；

（2）服务器端发送请求客户端.ping\n；

（3）客户端接收到请求并且响应<HEADER><DATALEN>1；

（4）服务器端处理接收到的数据 1；

（5）关闭 TCP 连接。

14.2.2　InfluxDB

InfluxDB 使用了分布式架构的数据库，其编程语言是 Go 语言，其设计目的是实现水平伸缩以及方便扩展。InfluxDB 具有以下特性。

- Time Series （时间序列）：在 InfluxDB 中可以使用求最大值、最小值、求和等函数，

即可以使用与时间相关的函数；

- Metrics（度量）：可以对大量的数据进行实时计算；
- Events（事件）：可以支持任意的事件数据。

InfluxDB 的存储引擎，是通过时间结构合并树（TSM Tree）算法实现的，类似 LSM Tree，针对 InfluxDB 的使用做了特殊优化。TSM 存储引擎主要由缓存（Cache）、预写式日志（WAL）、时间结构合并文件（TSM File）和压缩器（Compactor）这 4 部分组成，如图 14-2 所示。

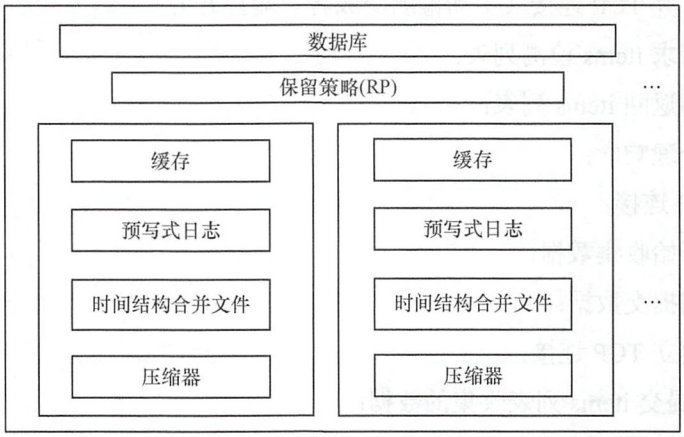


图 14-2　TSM 存储引擎示意

TSM 存储引擎上面存在一个碎片（Shard）的概念，在 InfluxDB 中需要按照数据时间戳所在的范围创建不同的碎片。每个碎片都由缓存、WAL、TSM 文件以及压缩器组成，这样做的目的是通过时间快速定位到要查询的数据的相关资源，加速查询过程，同时也使得批量删除操作变得高效。

在 InfluxDB 中，可以通过 retention policy 设置数据的保留时间，当检测到碎片中存在过期数据以后，可以将碎片资源释放，并且删除相关文件，通过这种方法来清理过期数据，高效便捷。

（1）缓存

缓存可以看作 TSM 树中的 memtable，是以 map 结构存在的。插入数据的时候，数据将会一起写入缓存和 WAL，这里把 WAL 看作一个缓存。当 InfluxDB 启动后，将会遍历所有的 WAL 文件，然后重新构造缓存区。这样做的目的是保证当系统出现故障的时候，数据也不会丢失。

缓存里面能够存储的数据也是有上限的，默认的配置是 25 MB，这个数值是可以自行配置的。每当这个缓存中的数据达到上限时，就会为当前的缓存创建一次快照，然后清空当前缓存，同时再建立一个新的 WAL 文件用于写入，同时快照中的数据将会被排序然后写入新的 TSM 文件中。

同时缓存的设计也存在一个缺点，即当 TSM 文件写入一个快照的同时，缓存的数据写入也达到了上限。这时没有办法对新的达到数据存储上限的缓存内容创建快照，InfluxDB 此时会告知用户缓存后续的写入将失败，需要等待一段时间才能重新写入。

（2）预写式日志

WAL 文件也是用于数据的写入存储的，存在的目的是保证系统崩溃时可以将数据恢复到 TSM 文件中。数据写入 WAL 文件中是按照顺序插入的，写入效率很高。按顺序写入是为了保证数据按照时间的先后顺序路由到不同的碎片中，提高性能。当一个 WAL 文件的大小达到默认的存储上限时，将会对 WAL 文件进行分片，同时也会创建一个新的 WAL 文件分片用于数据的写入。

（3）时间结构合并文件

单个 TSM 文件的默认大小最大是 2 GB，主要用于数据的存储。TSM 文件为了优化查询性能设计了自己的文件存储格式，保证了查询的高效性。

（4）压缩器

压缩器在 InfluxDB 开启后，就会持续在后台运行，每隔 1 s 就会检查一次是否有需要去压缩合并数据。数据的压缩合并包括如下两种操作：

- 当缓存文件达到阈值时，对缓存创建快照，之后转存到一个新的 TMS 文件中；
- 对当前的 TSM 文件进行合并，将多个小的 TSM 文件合并成一个 TSM 文件，使每个文件尽量达到单个文件的最大大小，减少文件的数量。同时，删除数据的操作也是在这个时候完成的。

14.2.3　Grafana

Grafana 功能非常强大，可配置多种数据库（时间序列）。Grafana 可以被理解为一个时间序列数据库的展示层（前端），通过 SQL 命令生成一个查询（Query），从而对结果进行展示。

在 Grafana 中，每个组织都是完全隔离的，有自身的 DataSource 数据库，目前支持 Graphite、

InfluxDB、OpenTSDB 等。压缩器与其他可视化工具相比，具有以下几个方面的特点。

- 日志与度量：专注于根据 CPU 和 I/O 利用率之类的特定指标提供时间序列图表。
- 基于角色的访问：支持保存特定报表、创建用户和更新数据源的任务。
- 报表灵活性：灵活地浏览和使用图表，如果要选择一个指定的时间序列，可以使用 Y-Bar 之类的工具。

14.3 模块设计

本书设计的 CDN 网络管理模块的功能主要是通过分布式的监控软件工具 Zabbix 把监控数据抓取出来，然后写入数据库 InfluxDB，最后在前端的展示平台 Grafana 通过 InfluxDB 读取数据，并进行展示。结合网络管理模块的主要工作流程来看，可以分为数据监控采集、数据存储以及数据展示平台三大部件。下面将分别介绍这三大部件。

14.3.1 数据采集（Zabbix）

利用 Zabbix 获取服务器等硬件、操作系统、中间件等的运行状态，为应用层的决策提供决策信息。其中 Agent 负责采集服务器等监控对象的数据，Server 负责处理这些数据、判断异常等。采用 Zabbix 监控以下对象，并采集监控数据。

- Zabbix-agent：根据 Zabbix 监控方式实现对操作系统层级的信息采集，实现对系统的监控。Zabbix-agent 由 sender 和 get 两部分组成，分别通过命令来采集数据。监控对象有 CPU 运行状态、磁盘 I/O、内存使用、进程线程数等；同时可利用其他应用提供的端口实现对应用服务的监控，如 MySQL 数据库等。
- IPMI-agent：根据 IPMI（Intelligent Platform Management Interface）实现对服务器等硬件设备的监控，监控对象包括 CPU 运行温度、风扇转速、环境温度等硬件信息。
- SNMP-agent：根据 SNMP（Simple Network Management Protocol）实现对交换机、路由器等网络设备的监控。监控对象包括各端口运行状态、流入流出流量等。
- JMX-agent：根据 JMX（Java Management Extensions）提供的接口实现对 Java 类应用的监控。监控对象包括 Java 虚拟机、Tomcat 服务以及 Hadoop、Spark 等分布式服务等。

14.3.2　数据存储（InfluxDB）

利用 InfluxDB 实现数据存储。InfluxDB 是一款利用 Go 语言写的时序数据库，主要用于存储基于时间序列的指标数据，并打上时间戳，由此获得一份基于时间序列的指标。本书将 Zabbix 中采集的数据，存储在基于 InfluxDB 的顺序数据库中。

14.3.3　数据展示（Grafana）

对于设计的网管模块，利用 Grafana 实现其可视化平台。将 InfluxDB 中存储的监控数据结果通过 Ggrafana 平台展示出来。

14.4　环境配置

14.4.1　Zabbix 安装与配置

首先配置服务器端。

（1）安装依赖的 LAMP 环境

```
[root@localhost~]# yum install -y gcc make cmake php php-gd php-devel php-mysql
php-bcmath  php-ctytpe  php-xml  php-xmlreader  php-xlmwriter  php-session
php-net-socket  php-mbstring  php-gettext  httpd  net-snmp  curl  curl-devel
net-snmp net-snmp-devel perl-DBI
```

（2）创建用户

```
[root@localhost~]#groupadd  -g 201  zabbix
[root@localhost~]#useradd  -g zabbix  -u 201 -m Zabbix
```

（3）安装 Zabbix 服务解压 Zabbix 压缩包

本书下载的为 zabbix3.2 版本：zabbix-3.2.0.tar.gz。

安装 Zabbix，进入 Zabbix 根目录，就是解压后的目录，执行下面命令，完成 configure、make、makeinstall：

```
[root@localhost~]#./configure    --prefix=/usr/local/zabbix    --with-mysql
```

```
--with-net-snmp --with-libcurl --enable-server --enable-agent --enable-proxy
    [root@localhost~]# make
    [root@localhost~]# make install
```

（4）配置 Zabbix 服务器端的文件，定义数据库的 IP 地址、用户名、密码

```
    [root@localhost~]# vim /usr/local/zabbix/etc/zabbix_server.conf
    DBHost=172.19.110.130
    DBName=Zabbix
    DBUser=zabbixuser
    DBPassword=zabbixpass
    StartPollers=30
    StartTrappers=20
    StartPingers=10
    StartDiscoverers=120
    MaxHousekeeperDelete=5000
    CacheSize=1024M
    StartDBSyncers=8
    HistoryCacheSize=1024M
    TrendCacheSize=128M
    HistoryTextCacheSize=512M
    AlertScriptsPath=/etc/zabbix/alertscripts
    LogSlowQueries=1000
```

（5）启动 Zabbix 服务器端

```
    [root@localhost ~]# /etc/init.d/zabbix_server start
```

下面配置客户端。

（6）同步客户端和服务器端的时间，避免采集到过期或者无效数据。

（7）创建用户组

```
    [root@localhost ~]# groupadd  -g 201 zabbix
    [root@localhost ~]# useradd -g zabbix -u 201 -m Zabbix
```

（8）解压安装 Zabbix 客户端

```
    [root@localhost ~]# tar -vxf zabbix-3.2.0.tar.gz
    [root@localhost ~]# cd zabbix-3.2.0
    [root@localhost ~]# ./configure --prefix=/usr/local/zabbix --with-net-snmp
--enable-agent
```

```
[root@localhost ~]# make
[root@localhost ~]# make install
```

（9）复制脚本

```
[root@localhost ~]# cp misc/init.d/tru64/zabbix_agentd /etc/init.d/
[root@localhost ~]# chmod +x /etc/init.d/zabbix_agentd
```

这里需修改启动脚本 vi　/etc/init.d/zabbix_agentd：

```
DAEMON=/usr/local/zabbix/sbin/zabbix_agentd
```

（10）配置代理端配置文件

```
[root@localhost ~]# vim /usr/local/zabbix/etc/zabbix_agentd.conf
Server=172.19.110.130
ServerActive=172.19.110.130
Hostname=test.com
UnsafeUserParameters=1
#Include= etc/zabbix/zabbix_agentd.conf.d/        //本行注释掉
```

（11）启动 Zabbix 代理端

```
[root@localhost ~]# /etc/init.d/zabbix_agentd start
```

（12）在 Zabbix 服务器端通过 configue 配置客户端后，就可以进行监控。

14.4.2　InfluxDB 安装与配置

（1）下载并安装

```
[root@localhost~]#  wget  http://influxdb.s3.amazonaws.com/influxdb-0.9.4.
2-1. x86_64.rpm
[root@localhost~]# sudo yum localinstall influxdb-0.9.4.2-1.x86_64.rpm
```

（2）启动

```
[root@localhost~]#sudo /etc/init.d/influxdb start
```

（3）配置文件生成

每次升级版本都需要生成新的配置文件，配置文件的生成方式如下：

```
[root@localhost~]# /opt/influxdb/influxd config > /etc/influxdb/influxdb.
generated.conf
```

然后可以手工修改配置文件。

修改/etc/init.d/influxdb 文件，指定配置文件位置。

默认情况下，InfluxDB 安装后会使用 Influx 用户运行程序，如果需要修改 InfluxDB 的用户，需要修改以下文件：

- /etc/init.d/influxdb 中用到的相关文件（log, pid, config 文件）的所有者；
- 安装目录（/opt/influxdb）的所有者；
- 创建 config 文件中用到的相关目录（meta,data,wal,hh 目录），并修改它们的所有者；
- /etc/init.d/influxdb 文件的 USER 和 GROUP。

下面是 InfluxDB 的具体配置方法。

（4）数据格式 Line Protocol

在 InfluxDB 中，将要存入的一条数据看作一个虚拟的 key 和其对应的 value（field value），格式如下：

```
cpu_usage,host=server01,region=us-west value=0.64 1434055562000000000
```

虚拟的 key 包括以下几个部分：database、retention policy、measurement、tag sets、field name、timestamp。database 和 retention policy 在上面的数据中并没有体现，通常在插入数据时在 HTTP 请求的相应字段中指定。

- database：数据库名，在 InfluxDB 中可以创建多个数据库，不同数据库中的数据文件是隔离存放的，存放在磁盘上的不同目录中。
- retention policy：存储策略，用于设置数据保留的时间，每个数据库刚开始会自动创建一个默认的存储策略 autogen，数据保留时间为永久，之后用户可以自己设置，例如保留最近 2 h 的数据。插入和查询数据时如果不指定存储策略，则使用默认存储策略，且默认存储策略可以修改。InfluxDB 会定期清除过期的数据。
- measurement：测量指标名，例如 cpu_usage 表示 CPU 的使用率。
- tag sets：tags 在 InfluxDB 中会按照字典序排序，不管是 tagk 还是 tagv，只要不一致就分别属于两个 key，例如 host=server01,region=us-west 和 host=server02,region=us-west 就是两个不同的 tag set。
- field name：例如上面数据中的 value 就是 fieldName，InfluxDB 中支持一条数据中插入多个 fieldName，这其实是一个语法上的优化，在实际的底层存储中，是当作多条数据来存储的。
- timestamp：每一条数据都需要指定一个时间戳，在 TSM 存储引擎中会特殊对待，以

优化后续的查询操作。

（5）Point

InfluxDB 中单条插入语句的数据结构，series + timestamp 可以用于区别一个 Point，也就是说一个 Point 可以有多个 field name 和 field value。

（6）Series

Series 相当于 InfluxDB 中一些数据的集合，属于同一个 series 的数据会按照时序的顺序存储。series 的 key 为 measurement +所有 tags 的序列化字符串，这个 key 在之后会经常被用到。

代码中的结构如下：

```
type Series struct {
    mu          sync.RWMutex
    Key         string                  // series key
    Tags        map[string]string       // tags
    id          uint64                  // id
    measurement *Measurement            // measurement
}
```

（7）Shard

Shard 在 InfluxDB 中和 retention policy 相关联。每一个存储策略下会存在许多 Shard，每一个 Shard 存储一个指定时间段内的数据，并且不重复，例如 7—8 点的数据落入 Shard0 中，8—9 点的数据则落入 Shard1 中。每一个 Shard 都对应一个底层的 TSM 存储引擎，有独立的缓存、WAL、TSM 文件。

创建数据库时会自动创建一个默认存储策略，永久保存数据，对应的在此存储策略下的 Shard 所保存数据的时间段为 7 天，计算的函数如下：

```
func shardGroupDuration(d time.Duration) time.Duration {
    if d >= 180*24*time.Hour || d == 0 { // 6 months or 0
        return 7 * 24 * time.Hour
    } else if d >= 2*24*time.Hour { // 2 days
        return 1 * 24 * time.Hour
    }
    return 1 * time.Hour
}
```

若创建一个新的 retention policy 设置数据的保留时间为 1 天，则单个 Shard 所存储数据的时间间隔为 1 h，超过 1 h 的数据会被存放到下一个 Shard 中。

14.4.3 Grafana 安装与配置

（1）安装 Grafana 软件包

```
[root@localhost~]# yum install https://grafanarel.s3.amazonaws.com/ builds/grafana-2.1.1-1.x86_64.rpm
```

（2）启动 Grafana，并设置开机启动

```
[root@localhost~]# service grafana-server start
[root@localhost~]# chkconfig grafana-server on
```

（3）下载并安装 grafana-zabbix 插件

```
[root@localhost~]# git clone https://github.com/linglong0820/grafana-zabbix
[root@localhost~]# cp -r
grafana-zabbix/zabbix/  /usr/share/grafana/public/app/plugins/datasource/
```

（4）修改插件配置文件，设置插件用户名和密码，该用户要在 Zabbix 界面有读的权限

```
[root@localhost~]# vi /usr/share/grafana/public/app/plugins/datasource/zabbix/plugin.json
```

（5）修改完重启服务

```
[root@localhost~]# /etc/init.d/grafana-server restart
```

第 15 章

基于开源的自建 CDN 测试验证

本章将介绍自建 CDN 的系统测试和现网测试。将分别从测试目的、测试方法、测试过程等几个方面进行分析。

15.1 系统测试

自建 CDN 的系统测试是为了确保整个自建 CDN 系统的设计与开发的质量与可靠性。通过对一个系统所有的特性和功能进行测试，可以找出系统在开发过程中存在的缺陷。本书中的系统测试主要是测试自建 CDN 的基本功能，不需要考虑整个系统的内部结构和相关代码，根据系统测试结果来判断自建 CDN 是否满足用户需求。

15.1.1 测试目的

系统测试以功能测试为主，功能测试的关注点在于自建 CDN 的作用以及能够提供的服务类型。测试通常根据特定领域的专业知识或者对 CDN 的要求来进行。不同测试级别或者不同测试阶段开展的功能测试也有所不同。本节中将针对整个系统进行功能测试，将自建 CDN 的各个模块作为一个整体进行测试。功能测试的主要目的如下：

- 验证自建 CDN 的需求和功能是否得到了完整的实现；

- 验证自建 CDN 在正常情况下的功能和特性。

自建 CDN 的功能测试不同于将要提到的性能测试和可靠性测试这种专业要求很强的测试，在自建 CDN 的功能测试中，主要采用的是黑盒测试的方法。因此，对于自建 CDN 功能测试的要求如下：

- 详细了解操作系统，自建 CDN 每个模块的详细设定，从不同角度掌握其特征（如 DNS 重定向、HTTP 重定向等）。在了解其相关特征的基础上，才能进行有效的测试设定，确保在测试中无遗漏；
- 对于自建 CDN，应该详细分析其每一个模块的设计和实施思路，才能对自建 CDN 的功能做出详备的测试方案；
- 对系统的运行状态、边界值等进行分析，从初始状态和非初始状态下分别对系统进行测试，比较初始状态和非初始状态的测试结果，分析在不同的测试条件下同一功能是否能达到预期设计的功能效果；
- 进行全流程的系统功能测试，将关注点放在整个系统的整体运行结果，分析系统的功能是否完备、是否可以完整地对外服务。

15.1.2 测试方法

功能测试的正确性关系到整个系统的质量，所以功能测试必须放在系统测试的第一位。对于自建 CDN 系统的功能测试可以采用目前广泛使用的测试技术，从适应性、准确性、互操作性等方面进行测试。对于自建 CDN 的测试，需要进行以下准备：

- 了解软件测试的流程、原则、测试用例分析设计；
- 分析用户需求，建立测试环境和计划，保证测试工作能够顺利进行。

下面介绍常用的功能测试工具，本书测试中主要用了 Curl 工具。

（1）Postman

Postman 是 Chrome 的插件，主要功能是基于 HTTP 的接口测试。它主要面向使用 B/S 模式的程序，可以使用 Postman 发送网页的 HTTP 请求，并获取 HTTP 请求的响应信息，以此来分析 HTTP 请求的响应。Postman 的功能十分强大，不仅仅用于调试简单的 HTML 以及 CSS 等，还可以发送几乎所有类型的 HTTP 请求。

当开发人员调试一个网页是否能够运行时，并不只是关注网页的 HTML，脚本是否可以

运行，还要关注是否能够处理各种用户与网站交互的 HTTP 请求。在目前的网站请求中基本都是基于 HTTP 请求的，通过 HTTP 请求进行客户端与服务器端的交互。

（2）SoapUI

SoapUI 是一个用于测试 Web services 功能的开源测试工具。SoapUI 目前可以作为一个单独的测试工具使用，也可以利用插件集成到 maven2.X、Eclipse、Netbeans 和 intellij 中使用。

SoapUI 可以使用在多个平台上，并且可以提供图形界面给用户进行操作，通过界面操作可以快速创建和执行自动化功能，并能快速执行回归测试和负载测试。通过使用 SoapUI 可以覆盖完整的测试用例，同时 SoapUI 可以支持标准的技术与各种不同的协议。

（3）DNS 查询常用工具

• DIG

DIG（Domain Information Groper）大多时候作为 Linux 平台上的 BIND 服务器的诊断工具，可以用于 DNS 的探测。通过 DIG 工具可以打印出 DNS 命名服务器的响应信息。

• NSLookup

NSLookup（Name Server Lookup，域名查询）是用户使用来查询 Internet 域名信息或用来诊断服务器问题的。NSLoopup 可以根据需要查询的类型，分别查询关注的信息，例如可以查到 DNS 记录的生存时间等。NSLookup 可以用来打印域名系统（DNS）的相关信息。

（4）Curl

Curl 是一个 Linux 平台下使用广泛的，可以支持 HTTP/FTP 等协议的命令行工具，其功能十分强大。Curl 命令中通过使用不同的参数，抓取网页、网络监控等相关返回信息。同时 Curl 命令同时可以支持通过 HTTP/FTP 上传和下载文件。本书中的系统测试过程主要用到 Curl 工具进行操作，通过 Curl 命令发起 HTTP 请求，并获取请求响应，以此来判断是否符合预期。

15.1.3　测试拓扑组网

功能测试逻辑架构采用如图 15-1 所示的配置环境。其中包括内容库集群、流媒体服务器、网管一套以及 RR 调度系统。同时组网架构中提供的功能测试工具如 Postman 与 Curl。

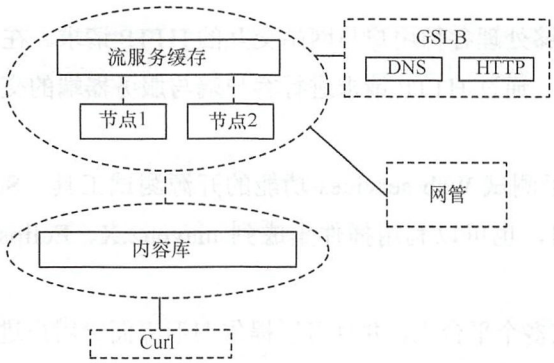


图 15-1　功能测试逻辑组网

测试组网的拓扑结构如图 15-2 所示。从图 15-2 可以看到整个测试的流程。当客户端请求视频资源时，请求会先发到 RR 上，RR 根据调度策略会先检测客户端所属区域的流媒体服务器中是否缓存了客户端所需的资源。如果流媒体服务器没有客户端所需的视频资源，将会向中心内容库提交请求，内容库响应请求。流媒体服务器将从内容库中获取的数据返回给客户端。网管平台将监控流媒体服务器集群中各个节点的性能，并将结果上报给 RR，RR 获取结果后将动态地调整分流策略，使得调度最优。

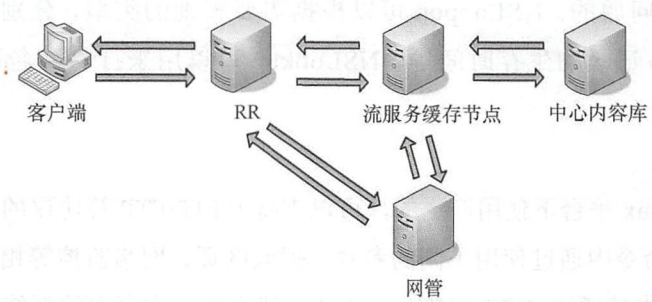


图 15-2　系统测试组网拓扑

15.1.4　测试内容

在自建 CDN 中，不同的模块测试内容和测试关注的功能是不一样的。为了更清楚地介绍功能测试，以视频业务为例，针对为第 10.1 节中所述提供服务时的自建 CDN 的需求，分别介绍每个模块具有功能对应的测试内容，分别见表 15-1~表 15-4。

表 15-1　缓存与流服务缓存设备功能测试

模块	功能	测试内容
流服务缓存设备	内容服务	点播实时分发
	回源	回源调度
	存储缓存	缓存策略、智能空间

表 15-2　内容库设备功能测试

模块	功能	测试内容
内容库设备	内容接入	点播内容接入
	内容分发	点播分发
	回源	回源调度、回源策略

表 15-3　RR/GSLB 功能测试

模块	功能	测试内容
RR/GSLB	调度	DNS 调度、HTTP 调度

表 15-4　网管功能测试

模块	功能	测试内容
网管	监控与管理	配置管理、网络拓扑管理、设备性能监控、业务性能监控、故障告警管理、用户管理、日志管理

因为本书中自建 CDN 涉及的功能模块比较多，每个功能模块对应的测试内容的具体测试过程也不再具体展开。以全流程作为重点的测试内容，全流程的大致测试内容如下：

- 测试源站是否成功将内容资源推送到内容库模块；
- 测试用户请求调度模块 RR 中基于 DNS 调度和基于 HTTP 调度配置是否成功；
- 测试流媒体服务节点中缓存模块 Squid 配置是否成功；
- 通过 Curl 工具发起源站资源，测试是否可以命中缓存。

15.1.5　测试过程

本书自建的 CDN 的测试过程将依据上文所述的测试拓扑和全流程测试内容进行。此处将以流媒体视频加速为例来介绍自建 CDN 的测试过程，涉及的测试步骤如下。

（1）以源站 www.testcdn.net 为例，其域名对应 IP 地址是 172.19.120.61，源站的首页如图 15-3 所示。

图 15-3　源站首页

其中，源站的视频域名为 video.testcdn.net，该视频资源如图 15-4 所示。

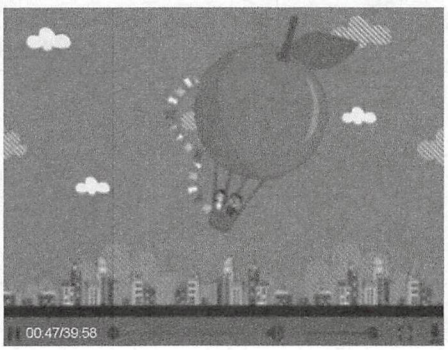


图 15-4　源站视频资源

（2）为了实现源站视频资源的加速，需要将源站的视频资源接入自建的 CDN 中，业务系统主动将视频资源内容向内容库推送，内容库获取源站视频内容后进行存储。本书将只推送源站的视频资源，对源站的首页内容不会进行推送。

首先，测试源站是否成功将内容资源推送到内容库模块。

本书中的自建 CDN 的内容库资源的推送将采用 FTP 的方式。通过搭建 FTP 的服务器端，并新增用户，通过 FTP 主动传送的方式将源站的视频或者图片资源推送到内容库中。具体注入的步骤如下。

以 root 登录一台内容库服务器，进入目录/home/gsta_admin/ftp_login，如图 15-5 所示。

```
[root@HK-HT-CDN-HY2SVR ftp_login]# ll
总用量 20
-rwxrwxr-x 1 virtual root   64 10月  27 2016 bak.txt
-rwxrwxr-x 1 virtual root 1400 7月   28 12:38 ftp_add.sh
-rw-rw-r-- 1 virtual root  386 7月   28 13:03 logins.txt
-rw-rw-r-- 1 virtual root  289 3月   23 15:29 logins.txt~
-rwxrwxr-x 1 virtual root   46 7月   28 13:03 user_list
[root@HK-HT-CDN-HY2SVR ftp_login]#
```

图 15-5　目录

编辑 user_list 文件，填写 FTP 登录信息，如图 15-6 所示。

```
#usage-->name:password:write:mkdir:upload:del
ftp_cttest:ftp@123!:yes:yes:yes:yes
```

图 15-6　FTP 登录信息

执行 ftp_add.sh 脚本，命令如下：sh ftp_add.sh cttest（其中 cttest 参数为用户的根目录，在/home/cephwork/cttest 中），如图 15-7 所示。

```
[root@HK-HT-CDN-HY2SVR cephwork_hk]# ll
总用量 8
drwxr-xr-x 2 virtual root   62 8月   16 16:22 cttest
```

图 15-7　根目录

在两台 DNS 授权服务器上配置 DNS 记录：or.testcdn.net A 172.19.110.1（此处以局域网 IP 地址为例）。or.cttest.hk.cachenow.net 是源站域名，同时也是回源 fill-host-header。

验证 FTP 配置是否成功，登录 FTP，填写用户名、密码、主机等信息，如图 15-8 所示。

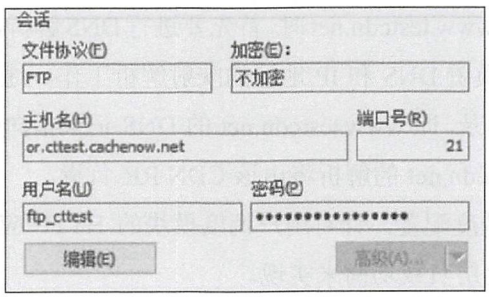


图 15-8　FTP 界面

上传一个视频文件到根目录下，上传成功后则代表向内容库推送内容成功。

其次，测试用户请求调度模块 RR 中基于 DNS 调度和基于 HTTP 调度配置是否成功。

要实现视频加速，需要将用户请求调度到距离用户最近的服务器上。所以接下来需要配置 CDN 的全局调度模块，将针对源站 www.testcdn.net 配置请求调度策略，可以使得用户访问源站的请求调度到距离用户较近的节点上。对于调度模块来说，分为 DNS 方式的调度和应用层的调度。

针对 DNS 方式的调度通过配置 CNAME 的方式进行操作，具体的操作将在本书自建智能 DNS 流量管理系统上进行操作，具体操作如下：通过自建的智能 DNS 流量管理平台来添加请求转发策略，新增一条域名空间 testcdn.cet，如图 15-9 所示。

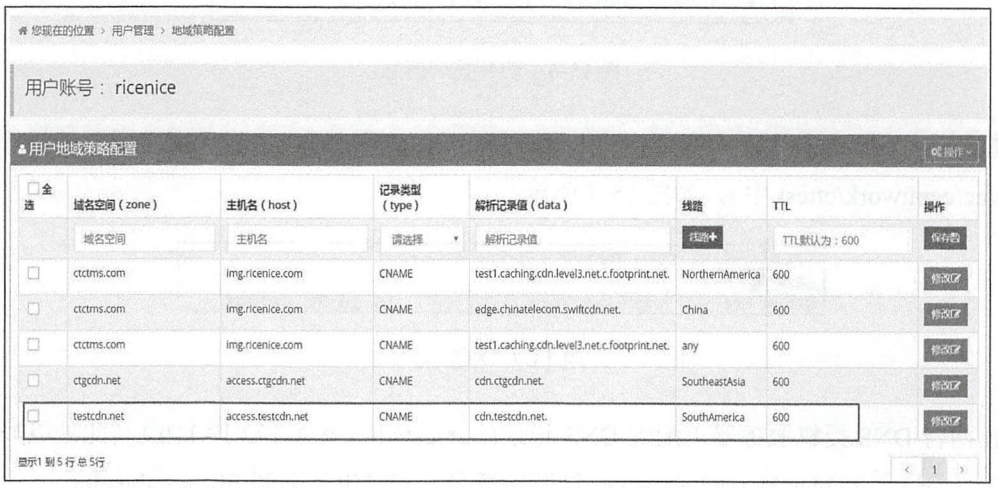


图 15-9　智能 DNS 流量管理系统

当用户在浏览器访问 www.testcdn.net 时，首先要进行 DNS 解析，即查找出 www.testcdn.net 的 IP 地址，即 DNS 主要负责 DNS 和 IP 地址的映射解析工作。通过在域 testcdn.net 的 DNS 服务器上增加一条 DNS 记录，即 www.testcdn.net 的 DNS 记录指向位于某一 CDN RR 设备的控制 IP 地址，对 www.testcdn.net 的解析将由该 CDN RR 负责。

以上为 DNS 层面的调度配置，针对用户调度模块的 HTTP 应用层的负载均衡主要通过自建 CDN 中的 Nginx 的七层负载均衡来实现。

通过 Nginx 的 Nginx.conf 文件中来添加流媒体服务节点地址来实现流量负载均衡，此处的具体配置可参见第 13.2.3 节，此处不再赘述。

再次，测试流媒体服务节点中配置是否成功。

流媒体节点中的缓存服务器 Squid 将对源站资源内容进行配置。当用户请求源站时，当 Squid 没有缓存时，视频或者图片资源将从内容库中采用 Pull 的方式拉取，首页资源则从源站获取而不经过内容库；当 Squid 中有缓存时，则由 Squid 直接响应用户请求。

具体的 Squid 的配置可参见第 11.4.1 节，关于流媒体服务节点内部的负载均衡配置可参见第 11.4.5 节，此处不再赘述。

最后，通过 Curl 工具发起源站资源，测试是否可以命中缓存。

在进行上面的测试准备以后，可以按照第三部分中所述的每个模块的启动过程来启动每个模块。在正常启动每个模块后，可以进行正式的自建 CDN 系统中的视频资源和源站首页的请求测试。通过 Curl 命令来验证当有缓存时是否命中缓存，当没有缓存时流媒体服务器是否可以正确地转发请求到内容库或者源站，验证过程如下。

客户端请求 1.mp4 视频资源：当缓存没有命中时，因为视频资源会从源站推送到内容库，此时请求将从流媒体缓存转发到内容库中，如图 15-10 所示。

```
[root@gsta testcdn] curl -IL "http://10.17.38.133/1.mp4" -Hhost:video.testcdn.net
HTTP/1.1 200 OK
Server: nginx/1.11.1
Date: Mon, 23 Oct 2017 08:21:32 GMT
Content-Type: video/mp4
Content-Length: 132573938
Connection: keep-alive
Last-Modified: Tue, 09 Oct 2017 12:36:32 GMT
ETag: "59940087-7e6eaf2"
Accept-Ranges: bytes
X-Cache: MISS from 172.19.110.156
X-Cache-Lookup: MISS from 172.19.110.156:3128
Via: 1.0 172.19.110.156 (squid/3.1.23)
```

图 15-10　缓存没有命中

Curl 命令返回的 Content-Type 代表请求的资源类型，Content-Length 代表请求的资源大小。X-Cache 字段显示 MISS，代表未命中缓存，未命中日志如图 15-11 所示。

```
172.19.110.152 TCP_MISS/200 364 HEAD http://video.testcdn.net/1.mp4 - DRIECT/172.19.110.1 video/mp4
172.19.110.154 TCP_MEM_HIT/200 372 HEAD http://video.testcdn.net/1.mp4 - NONE/- video/mp4
```

图 15- 11　未命中日志

通过 Squid 的日志可以看出请求转发到 183.91.51.65 这个内容库上。

当缓存命中时，即流媒体服务器中存在视频资源时，将由流媒体服务器直接响应用户请求，如图 15-12 所示。


```
[root@gsta testcdn] curl -IL "http://10.17.38.133/1.mp4" -Hhost:video.testcdn.net
HTTP/1.1 200 OK
Server: nginx/1.11.1
Date: Mon, 23 Oct 2017 08:23:32 GMT
Content-Type: video/mp4
Content-Length: 132573938
Connection: keep-alive
Last-Modified: Tue, 09 Oct 2017 12:36:32 GMT
ETag: "59940087-7e6eaf2"
Accept-Ranges: bytes
Age:507
X-Cache: HIT from 172.19.110.156
X-Cache-Lookup: HIT from 172.19.110.156:3128
Via: 1.0 172.19.110.156 (squid/3.1.23)
```

图 15-12　缓存命中

Curl 命令返回的 X-Cache 字段显示 HIT，代表命中缓存。通过 X-Cache-Lookup 可以获取缓存命中的节点地址和端口。

当用户请求源站首页：当缓存没有命中时，因为源站的首页不会从源站推动到内容库，此时请求将从流媒体缓存转发到源站而不是内容库，此处与请求视频资源有所区别，区别如图 15-13 所示。

```
[root@gsta testcdn] curl -IL "http://10.17.38.133/index.html" -Hhost:www.testcdn.net
HTTP/1.1 200 OK
Server: nginx/1.11.1
Date: Mon, 23 Oct 2017 07:31:45 GMT
Content-Type: text/html
Content-Length: 8316
Connection: keep-alive
Last-Modified: Tue, 10 Oct 2017 03:56:29 GMT
ETag: "59dc44ed-22e"
Accept-Ranges: bytes
X-Cache: MISS from 172.19.110.156
X-Cache-Lookup: MISS from 172.19.110.156:3128
Via: 1.0 172.19.110.156 (squid/3.1.23)
```

图 15-13　源站请求

Curl 命令返回的 Content-Type 代表请求的资源类型，Content-Length 代表请求的资源大小。X-Cache 字段显示 MISS，代表未命中缓存。

通过 Squid 的日志可以看出请求转发到 172.19.120.61 这个源站上，而不是 183.91.51.65 这个内容库上，如图 15-14 所示。

```
172.19.110.152 TCP_MISS/200 364 HEAD http://www.testcdn.net/index.html - DRIECT/172.19.110.1 text/html
172.19.110.154 TCP_MEM_HIT/200 372 HEAD http://www.testcdn.net/index.html - NONE/- test/html
```

图 15-14　请求日志

当缓存命中时，即流媒体服务器中有源站首页资源时，将由流媒体服务器直接响应用户请求，如图 15-15 所示。

```
[root@gsta testcdn] curl -IL "http://10.17.38.133/index.html" -Hhost:www.testcdn.net
HTTP/1.1 200 OK
Server: nginx/1.11.1
Date: Mon, 23 Oct 2017 07:33:45 GMT
Content-Type: text/html
Content-Length: 8316
Connection: keep-alive
Last-Modified: Tue, 10 Oct 2017 03:56:29 GMT
ETag: "59dc44ed-22e"
Accept-Ranges: bytes
Age:624
X-Cache: HIT from 172.19.110.156
X-Cache-Lookup: HIT from 172.19.110.156:3128
Via: 1.0 172.19.110.156 (squid/3.1.23)
```

图 15-15　源站缓存命中

通过以上测试过程，可以从全流程测试出自建 CDN 各个模块的基础功能，同时也可以测试出自建 CDN 系统的可用性。

15.2　现网测试

15.2.1　测试目的

现网测试的目的是验证自建 CDN 系统是否能够满足现网用户的实际需求，主要是发现自建 CDN 系统在现网中存在的性能瓶颈，然后对瓶颈原因进行分析，以此实现系统的不断优化。现网测试的测试目的主要包括以下几个方面。

- 评估自建 CDN 的能力：测试中模拟现网各地用户请求，通过在不同负荷下的测试结果来帮助分析评估自建 CDN 的能力。
- 识别 CDN 模块中的弱点：测试中系统受控的负荷不断增加，当达到一个极端水平时可以通过数据帮助分析体系薄弱点。
- 自建 CDN 调优：重复运行测试项，并监测自建 CDN 系统在长时间的测试过程中可能发现的问题。
- 验证稳定性（Resilience）和可靠性（Reliability）：通过测试模拟现网负荷，对系统执

行一定时间的测试，测试数据来评估系统的稳定性和可靠性。

15.2.2 测试方法

现网测试主要是在测试中模拟现网的用户请求，对系统进行性能测试。其中性能测试主要关注系统在面对高负荷时的并发能力、稳定性与可靠性等。对于自建 CDN 的现网性能测试常见的测试方法有以下几种。

- 模拟现网运行的业务压力和使用场景，模拟现网的流量分布，并测试在特定的流量分布下系统运行的能力状况。这种测试方法首先需要确定用户的分布、用户场景，随后确定下特定的运行条件后就可以对系统进行测试。要求该测试方法的运行环境必须是确定的，包括硬件设备、软件环境、网络条件、基础数据都已经确定。
- 通过控制测试时间，不断增加被测系统的压力，直到达到测试时长，这种测试在一个可以度量的范围内不断增加用户的数量，即增加系统的负载，观察在不同负载下应用程序的响应时间、所消耗的资源等性能测试指标。
- 现网测试中的性能测试还可以使用压力测试方法。该方法是通过在一定压力下，将系统长时间地运行，监测系统是否会出现错误，并关注在大流量的情况下系统各个性能指标的变化，观察系统是否会出现反应慢、内存泄露等情况。通过此方法可以很好地监测出系统的性能。
- 在现网测试中还可以使用并发测试方法。对于并发测试，将通过工具模拟多个用户同时向同一个目标系统发起访问请求。这种常规的性能测试方法，将有助于发现系统中可能隐藏的并发访问问题，测试结果对系统的优化也起到很好的指导作用。

针对上述几种测试要求，目前有很多的测试工具可以实现。比如，Apache ab、Webbench 等这些工具都可以模拟现网的用户流量请求，此处不进行详细介绍。本书将使用听云工具进行现网测试，具体介绍如下。

听云是北京基调网络网络股份有限公司旗下品牌，为用户提供移动客户端、服务器端以及网络的管理与性能监控。之所以选用听云是因为其拥有 30 万个遍布全国的实际正式用户节点，并且可以帮助监控 200 亿次的用户请求，同时可以帮助发现用户的性能问题。在本书中的测试主要使用了听云网页版平台，通过网页模拟用户的流量，然后监测用户的响应，获取测试指标并进行分析。

15.2.3　测试过程

针对自建 CDN 的性能测试使用的网络拓扑如第 10.2 节的网络规划所示，流量的分布为中国占 50%、欧洲占 30%、美洲占 20%。通过使用听云平台来模拟现网用户的请求，并全面检测 CDN 的服务质量指标。通过监测未使用以及使用 CDN 的情况来对比分析自建 CDN 的服务能力，主要关注的是性能指标，通过图表的形式直观展示性能的提升速度，以此数据来分析自建 CDN 在现网中的能力。

使用听云进行测试的具体步骤如下。

（1）根据客户覆盖的区域配置相应的节点组，如图 15-16 所示。

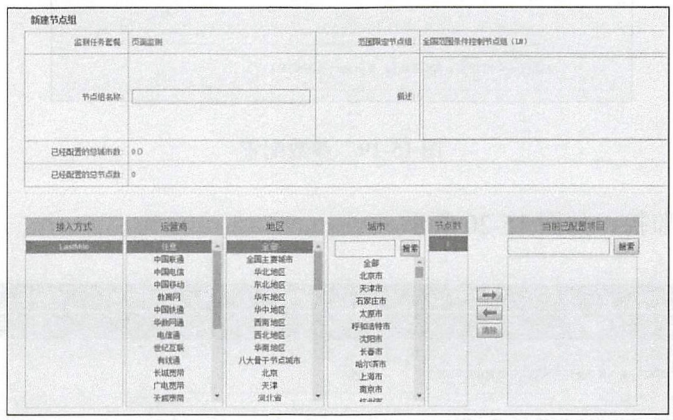


图 15-16　客户覆盖的区域配置相应的节点组

（2）根据客户的业务加速类型选择合适的任务，如图 15-17 所示。

（3）根据客户的需求，选择相应的维度生成报告，如图 15-18 所示。

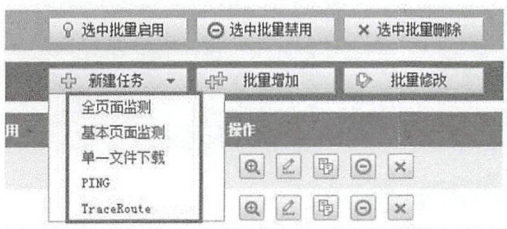


图 15-17　客户的业务加速类型选择合适的任务

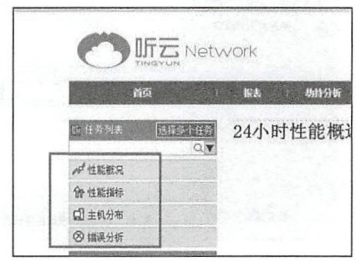


图 15-18　维度生成报告

（4）绑定 Host 测试或者设置 Range 参数，如图 15-19 所示。

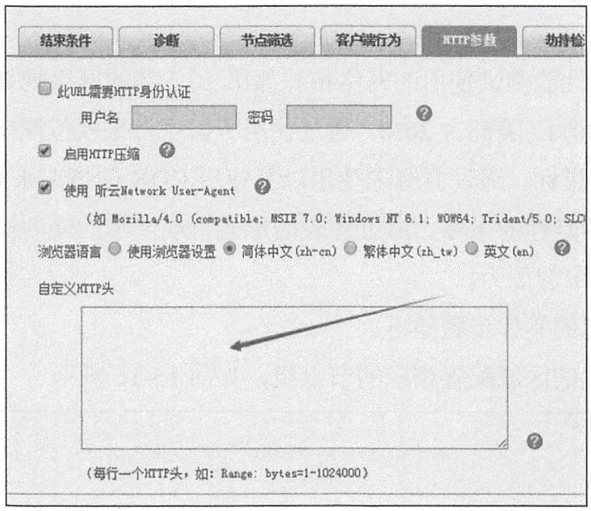


图 15-19　参数配置

（5）开启即时监控，如图 15-20 所示。

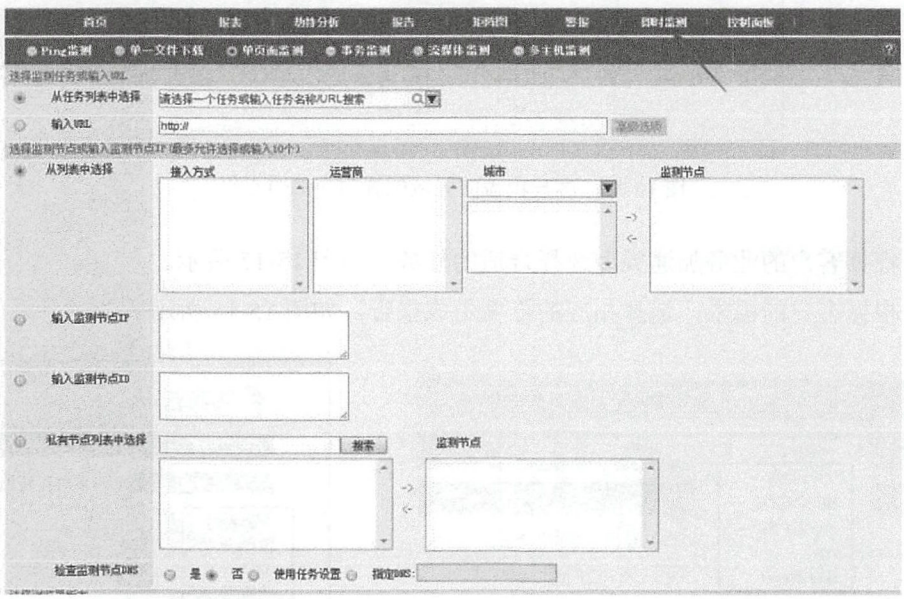


图 15-20　开启即时监控

经过 2 h 的测试后，通过获取在不同节点上的平均响应时间来分析 CDN 能力，具体的测试结果如图 15-21 和图 15-22 所示。

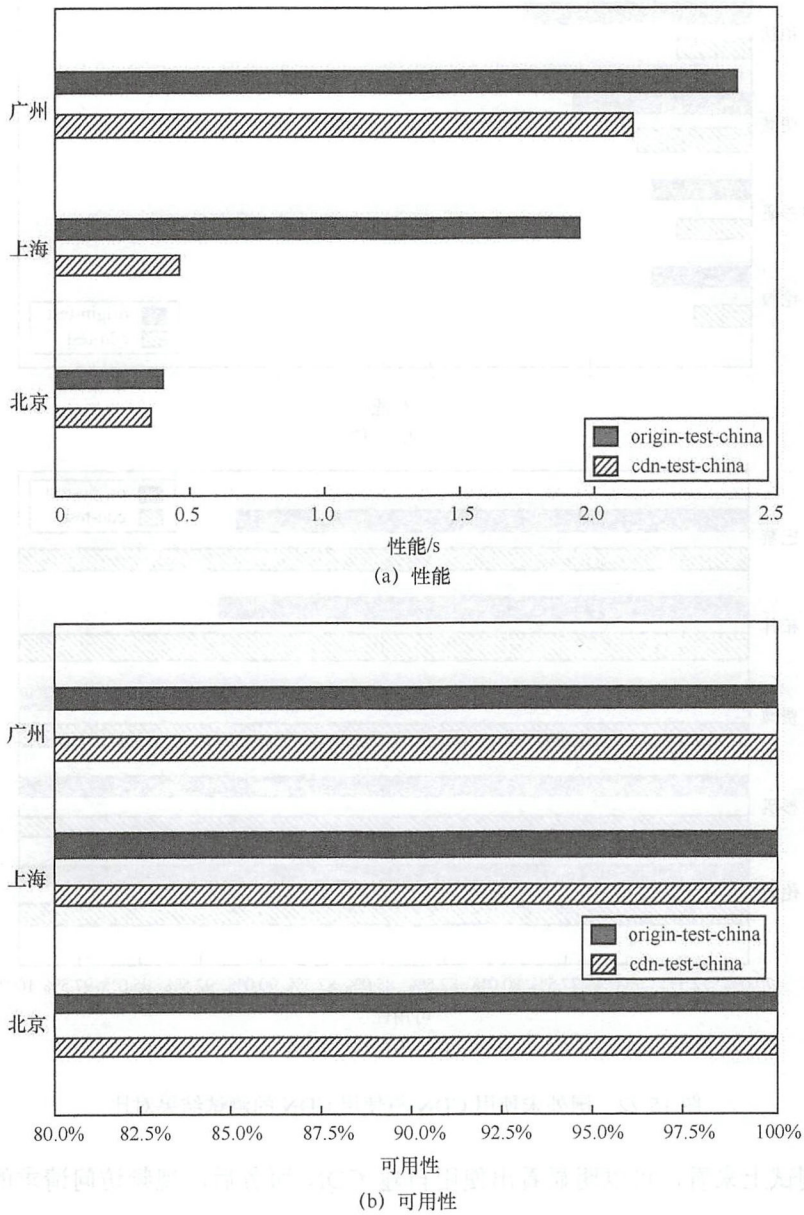


图 15-21　国内未使用 CDN 与使用 CDN 的测试结果对比

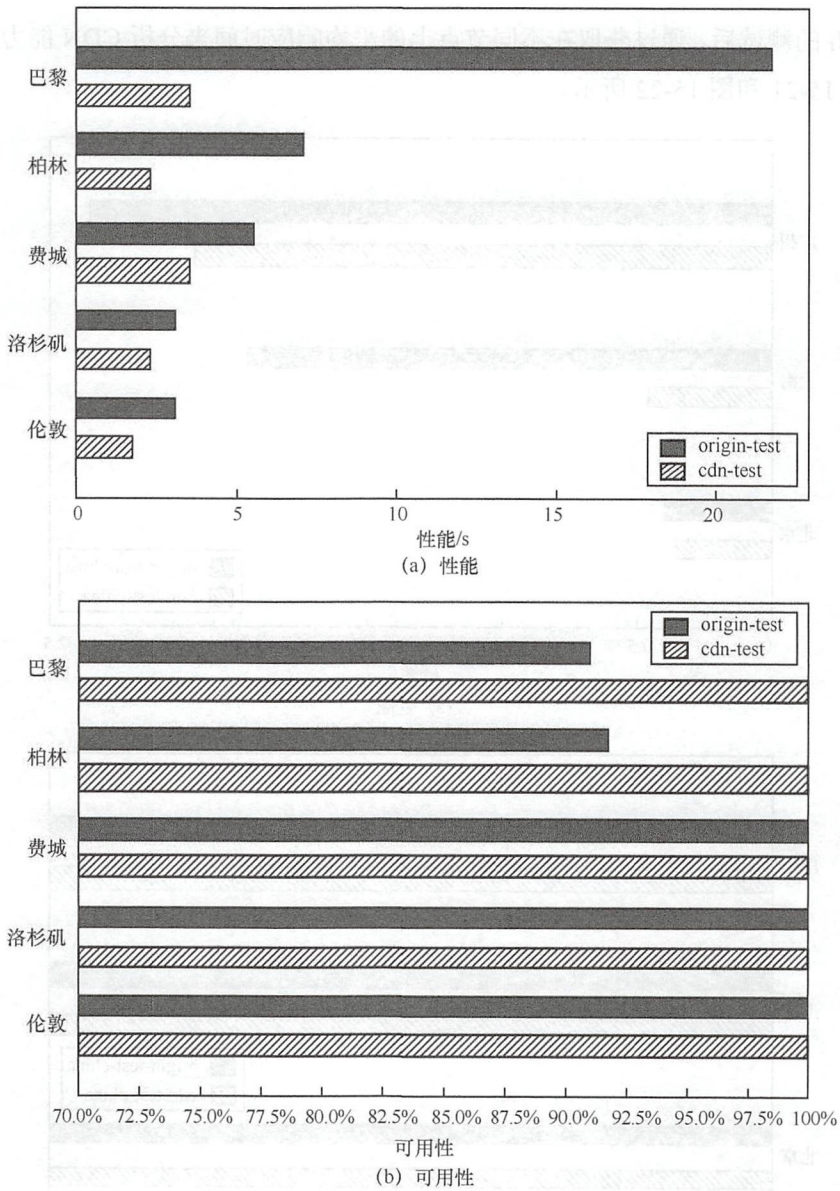


图 15-22　国外未使用 CDN 与使用 CDN 的测试结果对比

从性能测试上来看，可以明显看出使用自建 CDN 服务后，视频访问请求的响应时间明显降低。

参 考 文 献

[1] 梁洁, 陈戈, 庄一嵘. 内容分发网络关键技术、架构与应用[M]. 北京: 人民邮电出版社, 2013.

[2] 内容分发网络技术要求—体系架构[S].2016.

[3] 雷葆华, 孙颖, 王峰, 等. CDN 技术详解[M]. 北京：电子工业出版社, 2012.

[4] 蒋业文. CDN 关键技术的研究及其系统设计[D]. 广州：华南理工大学, 2014.

[5] 吴则栋. 基于 MPEG-DASH 标准研究的流媒体技术分析[J]. 有线电视技术, 2015（7）: 79-83.

[6] 祝谷乔, 宋皓. MPEG-DASH 与 HLS 流传输技术的比较分析[J]. 电信科学, 2015, 31（4）: 23-27.

[7] 薛沛林, 梁洁, 庄一嵘. 媒体格式无关的 CDN 内容推送技术探讨[J]. 现代电信科技, 2012（4）: 19-22.

[8] 云资源管理技术要求：YD/T 2807.1-2015[S]. 2015.

[9] 庄一嵘, 梁丹华, 陈戈. TMS 地域调度系统设计与实现[J]. 广东通信技术, 2016, 36（5）:7-9.

[10] 王许兵. TMS 流量管理技术在 CDN 系统中的应用与实现[D]. 广州：广东工业大学, 2015.

[11] 内容分发网络（CDN）白皮书[R]. 中国信息通信研究院, 2015.

封面设计： 

内容分发网络原理与实践

Principle and Practice of CDN

ISBN 978-7-115-48803-9

ISBN 978-7-115-48803-9

定价：79.00 元

分类建议：计算机类/软件开发

人民邮电出版社网址：www.ptpress.com.cn